目錄

安卓逆向系列教程

> 作者：飞龙

- 在线阅读
- PDF格式
- EPUB格式
- MOBI格式
- 代码仓库

更新历史

- v1.0：2017.4.4
    - 添加 Dalvik、APK、DEX 基础知识
    - 添加静态分析、抓包工具教程
    - 添加 12 个 APK 破解案例

赞助我

协议

安卓逆向系列教程（一）Dalvik 指令集

> 作者：飞龙

寄存器

Dalvik 指令集完全基于寄存器，也就是说，没有栈。

所有寄存器都是 32 位，无类型的。也就是说，虽然编译器会为每个局部变量分配一个寄存器，但是理论上一个寄存器中可以存放一个 `int` ，之后存放一个 `String` （的引用），之后再存放一个别的东西。

如果要处理 64 位的值，需要连续的两个寄存器，但是代码中仍然只写一个寄存器。这种情况下，你在代码中看到的 `vx` 实际上是指 `vx` 和 `vx + 1` 。

寄存器有两种命名方法。 `v` 命名法简单直接。假设一共分配了 10 个寄存器，那么我们可以用 `v0` 到 `v9` 来命名它们。



除此之外，还可以用 `p` 命名法来命名参数所用的寄存器，参数会占用后面的几个寄存器。假如上面那个方法是共有两个参数的静态方法，那么，我们就可以使用 `p0` 和 `p1` 取代 `v8` 和 `v9` 。如果是实例方法，那么可以用 `p0 ~ p2` 取代 `v7 ~ v9` ，其中 `p0` 是 `this` 引用。



但在实际的代码中，一般不会声明所有寄存器的数量，而是直接声明局部变量所用的寄存器（后面会看到）。也就是说局部变量和参数的寄存器是分开声明的。我们无需关心 `vx` 是不是 `py` ，只需知道所有寄存器的数量是局部变量与参数数量的和。

数据类型

Dalvik 拥有独特的数据类型表示方法，并且和 Java 类型一一对应：

Java 类型	Dalvik 表示
`boolean`	Z
`byte`	B
`short`	S
`char`	C
`int`	I
`long`	J
`float`	F
`double`	D
`void`	V
对象类型	L
数组类型	[

其中对象类型由 `L<包名>/<类名>;` （完全限定名称）表示，要注意末尾有个分号，比如 `String` 表示为 `Ljava/lang/String;` 。

数组类型是 `[` 加上元素类型，比如 `int[]` 表示为 `[I` 。左方括号的个数也就是数组的维数，比如 `int[][]` 表示为 `[[I` 。

类定义

一个 smali 文件中存放一个类，文件开头保存类的各种信息。类的定义是这样的。

```
.class <权限修饰符> <非权限修饰符> <完全限定名称>
.super <超类的完全限定名称>
.source <源文件名>
```

比如这是某个 `MainActivity` ：

```
.class public Lnet/flygon/myapplication/MainActivity;
.super Landroid/app/Activity;
.source "MainActivity.java"
```

我们可以看到该类是 `public` 的，完整名称是 `net.flygon.myapplication.MainActivity` ，继承了 `android.app.Activity` ，在源码中是 `MainActivity.java` 。如果类

是 `abstract` 或者 `final` 的，会在 `public/private/protected` 后面表示。

类可以实现接口，如果类实现了接口，那么这三条语句下面会出现 `.implements <接口的完全限定名称>` 。比如通常用于回调的匿名类中会出现 `.implements Landroid/view/View$OnClickListener;` 。

类还可以拥有注解，同样，这三条语句下方出现这样的代码：

```
.annotation <完全限定名称>
    键 = 值
    ...
.end annotation
```

这些语句下面就是类拥有的字段和方法。

字段定义

字段定义如下：

```
.field <权限修饰符> <非权限修饰符> <名称>:<类型>
```

其中非权限修饰符可以为 `final` 或者 `abstract` 。

比如我在 `MainActivity` 中定义一个按钮：

```
.field private button1:Landroid/widget/Button;
```

方法定义

方法定义如下：

```
.method <权限修饰符> <非权限修饰符> <名称>(<参数类型>)<返回值类型>
    ...
.end method
```

要注意如果有多个参数，参数之间是紧密挨着的，没有逗号也没有空格。如果某个方法的参数是 `int, int, String` ，那么应该表示为 `IILjava/lang/String;` 。

`.locals`

方法里面可以包含很多很多东西，可以说是反编译的重点。首先，方法开头处可能会含有局部变量个数声明和参数声明。`.locals <个数>` 可以用于变量个数声明，比如声明了 `.locals 10` 之后，我们就可以直接使用 `v0` 到 `v9` 的寄存器。

`.param`

另外，参数虽然也占用寄存器，但是声明是不在一起的。`.param px,"<名称>"` 用于声明参数。不知道是不是必需的。

`.prologue`

之后 `.prologue` 的下面是方法中的代码。代码是接下来要讲的东西。

`.line`

代码之间可能会出现 `.line <行号>` ，用来标识 Java 代码中对应的行，不过这个是非强制性的，修改之后对应不上也无所谓。

`.local`

还可能出现局部变量声明， `.local vx, "<名称>":<类型>` 。这个也是非强制性的，只是为了让你清楚哪些是具名变量，哪些是临时变量。临时变量没有这种声明，照样正常工作。甚至你把它改成不匹配的类型（ `int` 改成 `Object` ），也可以正常运行。

数据定义

指令	含义
const/4 vx,lit4	将 4 位字面值 `lit4` （扩展为 32 位）存入 `vx`
const/16 vx,lit16	将 16 位字面值 `lit16` （扩展为 32 位）存入 `vx`
const vx, lit32	将 32 位字面值 `lit32` 存入 `vx`
const-wide/16 vx, lit16	将 16 位字面值 `lit16` （扩展为 64 位）存入 `vx` 及 `vx + 1`
const-wide/32 vx, lit32	将 32 位字面值 `lit32` （扩展为 64 位）存入 `vx` 及 `vx + 1`
const-wide vx, lit64	将 64 位字面值 `lit64` 存入 `vx` 及 `vx + 1`
const/high16 v0, lit16	将 16 位字面值 `lit16` 存入 `vx` 的高位
const-wide/high16, lit16	将 16 位字面值 `lit16` 存入 `vx` 和 `vx + 1` 的高位
const-string vx, string	将指字符串常量（的引用） `string` 存入 `vx`
const-class vx, class	将指向类对象（的引用） `class` 存入 `vx`

这些指令会在我们给变量赋字面值的时候用到。下面我们来看看这些指令如何与 Java 代码对应，以下我定义了所有相关类型的变量。

```
boolean z = true;
z = false;
byte b = 1;
short s = 2;
int i = 3;
long l = 4;
float f = 0.1f;
double d = 0.2;
String str = "test";
Class c = Object.class;
```

编译之后的代码可能是这样：

```
const/4 v10, 0x1
const/4 v10, 0x0
const/4 v0, 0x1
const/4 v8, 0x2
const/4 v5, 0x3
const-wide/16 v6, 0x4
const v4, 0x3dcccccd    # 0.1f
const-wide v2, 0x3fc999999999999aL    # 0.2
const-string v9, "test"
const-class v1, Ljava/lang/Object;
```

我们可以看到，`boolean`、`byte`、`short`、`int` 都是使用 `const` 系列指令来加载的。我们在这里为其赋了比较小的值，所以它用了 `const/4`。如果我们选择一个更大的值，编译器会采用 `const/16` 或者 `const` 指令。然后我们可以看到 `const-wide/16` 用于为 `long` 赋值，说明 `const-wide` 系列指令用于处理 `long`。

接下来，`float` 使用 `const` 指令处理，`double` 使用 `const-wide` 指令处理。以 `float` 为例，它的 `const` 语句的字面值是 `0x3dcccccd`，比较费解。实际上它是保持二进制数据不变，将其表示为 `int` 得到的。

我们可以用这段 c 代码来验证。

```
int main() {
    int i = 0x3dcccccd;
    float f = *(float *)&i;
    printf("%f", f);
    return 0;
}
```

结果是 `0.100000`，的确是我们当初赋值的 0.1。

最后，`const-string` 用于加载字符串，`const-class` 用于加载类对象。虽然文档中写着“字符串的 ID”，但实际的反编译代码中是字符串字面值，比较方便。对于类对象来说，代码中出现的是完全先定名称。

数据移动

数据移动指令就是大名鼎鼎的 `move`：

指令	含义
move vx,vy	`vx = vy`
move/from16 vx,vy	`vx = vy`
move/16 vx,vy	`vx = vy`
move-wide vx,vy	`vx, vx + 1 = vy, vy + 1`
move-wide/from16 vx,vy	`vx, vx + 1 = vy, vy + 1`
move-wide/16 vx,vy	`vx, vx + 1 = vy, vy + 1`
move-object vx,vy	`vx = vy`
move-object/from16 vx,vy	`vx = vy`
move-object/16 vx,vy	`vx = vy`
move-result vx	将小于等于 32 位的基本类型（ `int` 等）的返回值赋给 `vx`
move-result-wide vx	将 `long` 和 `double` 类型的返回值赋给 `vx`
move-result-object vx	将对象类型的返回值（的引用）赋给 `vx`
move-exception vx	将异常对象（的引用）赋给 `vx` ，只能在 `throw` 之后使用

`move` 系列指令以及 `move-result` 用于处理小于等于 32 位的基本类型。 `move-wide` 系列指令和 `move-result-wide` 用于处理 `long` 和 `double` 类型。 `move-object` 系列指令和 `move-result-object` 用于处理对象引用。

另外不同后缀（无、 `/from16` 、 `/16` ）只影响字节码的位数和寄存器的范围，不影响指令的逻辑。

数据运算

二元运算

二元运算指令格式为 `<运算类型>-<数据类型> vx,vy,vz` 。其中算术运算的 `type` 可以为 `int` 、 `long` 、 `float` 、 `double` 四种（ `short` 、 `byte` 按 `int` 处理），位运算的只支持 `int` 、 `long` ，下同。

指令	运算类型	含义	
算术运算			
add- vx, vy, vz	加法	vx = vy + vz	
sub- vx, vy, vz	减法	vx = vy - vz	
mul- vx, vy, vz	乘法	vx = vy * vz	
div- vx, vy, vz	除法	vx = vy / vz	
rem- vx, vy, vz	取余	vx = vy % vz	
位运算			
and- vx, vy, vz	与	vx = vy & vz	
or- vx, vy, vz	或	`vx = vy	vz`
xor- vx, vy, vz	异或	vx = vy ^ vz	
shl- vx, vy, vz	左移	vx = vy << vz	
shr- vx, vy, vz	算术右移	vx = vy >> vz	
ushr- vx, vy, vz	逻辑右移	vx = vy >>> vz	

我们可以查看如下代码：

```
int a = 5,
    b = 2,
    c = a + b,
    d = a - b,
    e = a * b,
    f = a / b,
    g = a % b,
    h = a & b,
    i = a | b,
    j = a ^ b,
    k = a << b,
    l = a >> b,
    m = a >>> b;
```

编译后的代码可能为：

```
const/4 v0, 0x5
const/4 v1, 0x2
add-int v2, v0, v1
sub-int v3, v0, v1
mul-int v4, v0, v1
div-int v5, v0, v1
rem-int v6, v0, v1
and-int v7, v0, v1
or-int v8, v0, v1
xor-int v9, v0, v1
shl-int v10, v0, v1
shr-int v11, v0, v1
ushr-int v12, v0, v1
```

这里有个特例，当操作数类型是 `int` ，并且第二个操作数是字面值的时候，有一组特化的指令：

指令	运算类型	含义
算术运算		
add-int/ vx, vy,	加法	vx = vy + <litn>
sub-int/ vx, vy,	减法	vx = vy - <litn>
mul-int/ vx, vy,	乘法	vx = vy * <litn>
div-int/ vx, vy,	除法	vx = vy / <litn>
rem-int/ vx, vy,	取余	vx = vy % <litn>
位运算		
and-int/ vx, vy,	与	vx = vy & <litn>
or-int/ vx, vy,	或	`vx = vy `
xor-int/ vx, vy,	异或	vx = vy ^ <litn>
shl-int/ vx, vy,	左移	vx = vy << <litn>
shr-int/ vx, vy,	算术右移	vx = vy >> <litn>
ushr-int/ vx, vy,	逻辑右移	vx = vy >>> <litn>

其中 <litn> 可以为 `lit8` 或 `lit16` ，即 8 位或 16 位的整数字面值。比如 `int a = 0; a += 2;` 可能编译为 `const/4 v0, 0` 和 `add-int/lit8 v0, v0, 0x2` 。

二元运算赋值

二元运算赋值指令格式为 <运算类型>-<数据类型>/2 vx,vy,vz 。

指令	运算类型	含义	
算术运算			
add-/2addr vx, vy	加法赋值	vx += vy	
sub-/2addr vx, vy	减法赋值	vx -= vy	
mul-/2addr vx, vy	乘法赋值	vx *= vy	
div-/2addr vx, vy	除法赋值	vx /= vy	
rem-/2addr vx, vy	取余赋值	vx %= vy	
位运算			
and-/2addr vx, vy	与赋值	vx &= vy	
or-/2addr vx, vy	或赋值	`vx	= vy`
xor-/2addr vx, vy	异或赋值	vx ^= vy	
shl-/2addr vx, vy	左移赋值	vx <<= vy	
shr-/2addr vx, vy	算术右移赋值	vx >>= vy	
ushr-/2addr vx, vy	逻辑右移赋值	vx >>>= vy	

我们可以查看这段代码：

```
int a = 5,
    b = 2;
a += b;
a -= b;
a *= b;
a /= b;
a %= b;
a &= b;
a |= b;
a ^= b;
a <<= b;
a >>= b;
a >>>= b;
```

可能会编译成：

```
const/4 v0, 0x5
const/4 v1, 0x2
add-int/2addr v0, v1
sub-int/2addr v0, v1
mul-int/2addr v0, v1
div-int/2addr v0, v1
rem-int/2addr v0, v1
and-int/2addr v0, v1
or-int/2addr v0, v1
xor-int/2addr v0, v1
shl-int/2addr v0, v1
shr-int/2addr v0, v1
ushr-int/2addr v0, v1
```

一元运算

指令	运算类型	含义
算术运算		
neg- vx, vy	取负	`vx = -vy`
位运算		
not- vx, vy	取补	`vx = ~vy`

简单来说，如果代码为 `int a = 5, b = -a, c = ~a` ，并且变量依次分配给 `v0, v1, v2` 的话，我们会得到 `const/4 v0, 0x5` 、 `neg-int v1, v0` 和 `not-int v2, v0` 。

跳转

无条件

Java 里面没有 `goto` ，但是 Smali 里面有，一般来说和 `if` 以及 `for` 配合的可能性很大，还有一个作用就是用于代码混淆。

指令	类型
goto target	8 位无条件跳
goto/16 target	16 位无条件跳
goto/32 target	32 位无条件跳

`target` 在 Smali 中是标签，以冒号开头，使用方式是这样：

```
goto :label

# 一些语句

:label
```

这三个指令在使用形式上都一样，就是位数越大的语句支持的距离也越长。

条件跳转

`if` 系列指令可用于 `int` （以及 `short` 、 `char` 、 `byte` 、 `boolean` 甚至是对象引用）：

指令	含义
if-eq vx,vy,target	`vx == vy` 则跳到 target
if-ne vx,vy,target	`vx != vy` 则跳到 target
if-lt vx,vy,target	`vx < vy` 则跳到 target
if-ge vx,vy,target	`vx >= vy` 则跳到 target
if-gt vx,vy,target	`vx > vy` 则跳到 target
if-le vx,vy,target	`vx <= vy` 则跳到 target
if-eqz vx,target	`vx == 0` 则跳到 target
if-nez vx,target	`vx != 0` 则跳到 target
if-ltz vx,target	`vx < 0` 则跳到 target
if-gez vx,target	`vx >= 0` 则跳到 target
if-gtz vx,target	`vx > 0` 则跳到 target
if-lez vx,target	`vx <= 0` 则跳到 target

看一下这段代码：

```
int a = 10
if(a > 0)
    a = 1;
else
    a = 0;
```

可能的编译结果是：

```
const/4 v0, 0xa
if-lez v0, :cond_0 # if 块开始
const/4 v0, 0x1
goto :cond_1       # if 块结束
:cond_0            # else 块开始
const/4 v0, 0x0
:cond_1            # else 块结束
```

我们会看到用于比较逻辑是反着的，Java 里是大于，Smali 中就变成了小于等于，这个要注意。也有一些情况下，逻辑不是反着的，但是 `if` 块和 `else` 块会对调。还有，标签不一定是一样的，后面的数字会变，但是多数情况下都是两个标签，一个相对跳一个绝对跳。

如果只有 `if` ：

```
int a = 10;
if(a > 0)
    a = 1;
```

相对来说就简单一些，只需要在条件不满足时跳过 `if` 块即可：

```
const/4 v0, 0xa
if-lez v0, :cond_0 # if 块开始
const/4 v0, 0x1
:cond_0            # if 块结束
```

比较

对于 `long` 、 `float` 和 `double` 又该如何比较呢？Dalvik 提供了下面这些指令：

指令	含义
cmpl-float vx, vy, vz	`vx = -sgn(vy - vz)`
cmpg-float vx, vy, vz	`vx = sgn(vy - vz)`
cmp-float vx, vy, vz	`cmpg-float` 的别名
cmpl-double vx, vy, vz	`vx = -sgn(vy - vz)`
cmpg-double vx, vy, vz	`vx = sgn(vy - vz)`
cmp-double vx, vy, vz	`cmpg-double` 的别名
cmp-long vx, vy, vz	`vx = sgn(vy - vz)`

其中 `sgn(x)` 是符号函数，定义为：`x > 0` 时值为 1，`x = 0` 时值为 0，`x < 0` 时值为 -1。

我们把之前例子中的 `int` 改为 `float`：

```
float a = 10;
if(a > 0)
    a = 1;
else
    a = 0;
```

我们会得到：

```
const v0, 0x41200000 # float 10
const v1, 0x0
cmp-float v2, v0, v1
if-lez v2, :cond_0   # if 块开始
const v0, 0x3f800000 # float 1
goto :goto_0         # if 块结束
:cond_0              # else 块开始
const/4 v0, 0x0
:goto_0              # else 块结束
```

由于 `cmpg` 更类似平时使用的比较器，用起来更加顺手，但是 `cmpl` 也需要了解。

`switch`

Dalvik 共支持两种 `switch`，密集和稀疏。先来看密集 `switch`，密集的意思是 `case` 的序号是挨着的：

```
int a = 10;
switch (a){
    case 0:
        a = 1;
        break;
    case 1:
        a = 5;
        break;
    case 2:
        a = 10;
        break;
    case 3:
        a = 20;
        break;
}
```

编译为：

```
const/16 v0, 0xa

packed-switch v0, :pswitch_data_0 # switch 开始

:pswitch_0                        # case 0
const/4 v0, 0x1
goto :goto_0

:pswitch_1                        # case 1
const/4 v0, 0x5
goto :goto_0

:pswitch_2                        # case 2
const/16 v0, 0xa
goto :goto_0

:pswitch_3                        # case 3
const/16 v0, 0x14
goto :goto_0

:goto_0                           # switch 结束
return-void

:pswitch_data_0                   # 跳转表开始
.packed-switch 0x0                # 从 0 开始
    :pswitch_0
    :pswitch_1
    :pswitch_2
    :pswitch_3
.end packed-switch                # 跳转表结束
```

然后是稀疏 `switch`：

```
int a = 10;
switch (a){
    case 0:
        a = 1;
        break;
    case 10:
        a = 5;
        break;
    case 20:
        a = 10;
        break;
    case 30:
        a = 20;
        break;
}
```

编译为：

```
const/16 v0, 0xa

sparse-switch v0, :sswitch_data_0 # switch 开始

:sswitch_0                        # case 0
const/4 v0, 0x1
goto :goto_0

:sswitch_1                        # case 10
const/4 v0, 0x5

goto :goto_0

:sswitch_2                        # case 20
const/16 v0, 0xa
goto :goto_0

:sswitch_3                        # case 15
const/16 v0, 0x14
goto :goto_0

:goto_0                           # switch 结束
return-void

.line 55
:sswitch_data_0                   # 跳转表开始
.sparse-switch
    0x0 -> :sswitch_0
    0xa -> :sswitch_1
    0x14 -> :sswitch_2
    0x1e -> :sswitch_3
.end sparse-switch                # 跳转表结束
```

数组操作

数组拥有一套特化的指令。

创建

指令	含义
new-array vx,vy,type	创建类型为 `type` ，大小为 `vy` 的数组赋给 `vx`
filled-new-array {params},type_id	从 `params` 创建数组，结果使用 `move-result` 获取
filled-new-array-range {vx..vy},type_id	从 `vx` 与 `vy` 之间（包含）的所有寄存器创建数组，结果使用 `move-result` 获取

对于第一条指令，如果我们这样写：

```
int[] arr = new int[10];
```

就可以使用该指令编译：

```
const/4 v1, 0xa
new-array v0, v1, I
```

但如果我们直接使用数组字面值给一个数组赋值：

```
int[] arr = {1, 2, 3, 4, 5};
// 或者
arr = new int[]{1, 2, 3, 4, 5};
```

可以使用第二条指令编写如下：

```
const/4 v1, 0x1
const/4 v2, 0x2
const/4 v3, 0x3
const/4 v4, 0x4
const/4 v5, 0x5
filled-new-array {v1, v2, v3, v4, v5}, I
move-result v0
```

我们这里的寄存器是连续的，实际上不一定是这样，如果寄存器是连续的，还可以改写为第三条指令：

```
const/4 v1, 0x1
const/4 v2, 0x2
const/4 v3, 0x3
const/4 v4, 0x4
const/4 v5, 0x5
filled-new-array-range {v1..v5}, I
move-result v0
```

元素操作

`aget` 系列指令用于读取数组元素，效果为 `vx = vy[vz]` ：

```
aget vx,vy,vz
aget-wide vx,vy,vz
aget-object vx,vy,vz
aget-boolean vx,vy,vz
aget-byte vx,vy,vz
aget-char vx,vy,vz
aget-short vx,vy,vz
```

有两个指令需要说明， `aget` 用于获取 `int` 和 `float` ， `aget-wide` 用于获取 `long` 和 `double` 。

同样， `aput` 系列指令用于写入数组元素，效果为 `vy[vz] = vx` ：

```
aget vx,vy,vz
aget-wide vx,vy,vz
aget-object vx,vy,vz
aget-boolean vx,vy,vz
aget-byte vx,vy,vz
aget-char vx,vy,vz
aget-short vx,vy,vz
```

如果我们编写以下代码：

```
int[] arr = new int[2];
int b = arr[0];
arr[1] = b;
```

可能会编译成：

```
const/4 v0, 0x2
new-array v1, v0, I
const/4 v0, 0x0
aget-int v2, v1, v0
const/4 v0, 0x1
aput-int v2, v1, v0
```

对象操作

对象创建

指令	含义
new-instance vx, type	创建`type`的新实例，并赋给`vx`

`new-instance`用于创建实例，但之后还需要调用构造器`<init>`，比如：

```
Object obj = new Object();
```

会编译成：

```
new-instance v0, Ljava/lang/Object;
invoke-direct-empty {v0}, Ljava/lang/Object;-><init>()V
```

方法调用后面再讲。

字段操作

`sget`系列指令用于获取静态字段，效果为`vx = class.field`：

```
sget vx, type->field:field_type
sget-wide vx, type->field:field_type
sget-object vx, type->field:field_type
sget-boolean vx, type->field:field_type
sget-byte vx, type->field:field_type
sget-char vx, type->field:field_type
sget-short vx, type->field:field_type
```

`sput`系列指令用于设置静态字段，效果为`class.field = vx`：

```
sput vx, type->field:field_type
sput-wide vx, type->field:field_type
sput-object vx, type->field:field_type
sput-boolean vx, type->field:field_type
sput-byte vx, type->field:field_type
sput-char vx, type->field:field_type
sput-short vx, type->field:field_type
```

我们在这里创建一个类：

```
public class Test
{
    private static int staticField;

    public static int getStaticField() {
        return staticField;
    }

    public static void setStaticField(int staticField) {
        Test.staticField = staticField;
    }
}
```

编译之后，我们可以在 `getStaticField` 中找到：

```
sget v0, Lnet/flygon/myapplication/Test;->staticField:I
return v0
```

在 `setStaticField` 中可以找到：

```
sput p0, Lnet/flygon/myapplication/Test;->staticField:I
return-void
```

`iget` 系列指令用于获取实例字段，效果为 `vx = vy.field`：

```
iget vx, vy, type->field:field_type
iget-wide vx, vy, type->field:field_type
iget-object vx, vy, type->field:field_type
iget-boolean vx, vy, type->field:field_type
iget-byte vx, vy, type->field:field_type
iget-char vx, vy, type->field:field_type
iget-short vx, vy, type->field:field_type
```

`iput` 系列指令用于设置实例字段，效果为 `vy.field = vx`：

```
iput vx, vy, type->field:field_type
iput-wide vx, vy, type->field:field_type
iput-object vx, vy, type->field:field_type
iput-boolean vx, vy, type->field:field_type
iput-byte vx, vy, type->field:field_type
iput-char vx, vy, type->field:field_type
iput-short vx, vy, type->field:field_type
```

我们将之前的类修改一下：

```
public class Test
{
    private int instanceField;

    public int getInstanceField() {
        return instanceField;
    }

    public void setInstanceField(int instanceField) {
       this.instanceField = instanceField;
    }
}
```

反编译之后，我们可以在 `getInstanceField` 中找到：·

```
iget v0, p0, Lnet/flygon/myapplication/Test;->instanceField:I
return v0
```

在 `setInstanceField` 中可以找到：

```
iset p1, p0, Lnet/flygon/myapplication/Test;->instanceField:I
return-void
```

在实例方法中，`this` 引用永远是 `p0` 。第一个参数从 `p1` 开始。

方法调用

有五类方法调用指令：

指令	含义
invoke-static	调用静态方法
invoke-direct	调用直接方法
invoke-direct-empty	无参的 `invoke-direct`
invoke-virtual	调用虚方法
invoke-super	调用超类的虚方法
invoke-interface	调用接口方法

这些指令的格式均为：

```
invoke-* {params}, type->method(params_type)return_type
```

如果需要传递 `this` 引用，将其放置在 `param` 的第一个位置。

那么这些指令有什么不同呢？首先要分辨两个概念，虚方法和直接方法（JVM 里面叫特殊方法）。其实 Java 是没有虚方法这个概念的，但是 DVM 里面有，直接方法是指类的（ `type` 为某个类）所有实例构造器和 `private` 实例方法。反之 `protected` 或者 `public` 方法都叫做虚方法。

`invoke-static` 比较好分辨，当且仅当调用静态方法时，才会使用它。

`invoke-direct` （在 JVM 中叫做 `invokespecial` ）用于调用直接方法， `invoke-virtual` 用于调用虚方法。除了一种情况，显式使用 `super` 调用超类的虚方法时，使用 `invoke-super` （直接方法仍然使用 `invoke-direct` ）。

就比如说，每个 `Activity` 的 `onCreate` 中要调用 `super.onCreate` ，该方法属于虚方法，于是我们会看到：

```
invoke-super {p0, p1}, Landroid/app/Activity;->onCreate(Landroid
/os/Bundle;)V
```

但是呢，每个 `Activity` 构造器里面要调用 `super` 的无参构造器，它属于直接方法，那么我们会看到：

```
invoke-direct {p0}, Landroid/app/Activity;-><init>()V
```

`invoke-interface` 用于调用接口方法，接口方法就是接口的方法， `type` 一定为某个接口，而不是类。换句话说，类中实现的方法仍然是虚方法。比如我们在某个对象上调用 `Map.get` ，属于接口方法，但是调用 `HashMap.get` ，属于虚方法。这个指令一般在向上转型为接口类型的时候出现。

此外，五类指令中每一个都有对应的 `invoke-*-range` 指令，格式为：

```
invoke-*-range {vx..vy},type->method(params_type)return_type
```

如果参数所在的寄存器的连续的，可以替换为这条指令。

对象转换

对象转换有自己的一套检测方式，DVM 使用以下指令来实现：

指令	含义
instance-of vx, vy, type	检验 `vy` 的类型是不是 `type` ，将结果存入 `vx`
check-cast vx, type	检验 vx 类型是不是 type ，不是的话会抛出 ClassCastException

`instance-of` 指令对应 Java 的 `instanceof` 运算符。如果我们编写：

```
String s = "test";
boolean b = s instanceof String;
```

可能会编译为：

```
const-string v0, "test"
instance-of v1, v0, Ljava/lang/String;
```

`check-cast` 用于对象类型强制转换的情况，如果我们编写：

```
String s = "test";
Object o = (Object)s;
```

那么就会：

```
const-string v0, "test"
check-cast v0, Ljava/lang/Object;
move-object v1, v0
```

返回

```
return-void
return vx
return-wide vx
return-object vx
```

如果函数无返回值，那么使用 `return-void` ，注意在 Java 中，无返回值函数结尾处的 `return` 可以省，而 Smali 不可以。

如果函数需要返回对象，使用 `return-object` ；需要返回 `long` 或者 `double` ，使用 `return-wide` ；除此之外所有情况都使用 `return` 。

异常指令

异常指令实际上只有一条，但是代码结构相当复杂。

指令	含义
throw vx	抛出 `vx` （所指向的对象）

我们需要看看 Smali 如何处理异常。

try-catch

不失一般性，我们构造以下语句：

```
int a = 10;
try {
    callSomeMethod();
} catch (Exception e) {
    a = 0;
}
callAnotherMethod();
```

可能会编译成这样，这些语句每个都不一样，可以按照特征来定位：

```
const/16 v0, 0xa

:try_start_0            # try 块开始
invoke-direct {p0}, Lnet/flygon/myapplication/SubActivity;->call
SomeMethod()V
:try_end_0              # try 块结束

.catch Ljava/lang/Exception; {:try_start_0 .. :try_end_0} :catch
_0

:goto_0
invoke-direct {p0}, Lnet/flygon/myapplication/SubActivity;->call
AnotherMethod()V
return-void

:catch_0                # catch 块开始
move-exception v1
const/4 v0, 0x0
goto :goto_0            # catch 块结束
```

我们可以看到，`:try_start_0` 和 `:try_end_0` 之间的语句如果存在异常，则会向下寻找 `.catch`（或者 `.catch-all`）语句，符合条件时跳到标签的位置，这里是 `:catch_0`，结束之后会有个 `goto` 跳回去。

try-finally

```
int a = 10;
try {
    callSomeMethod();
} finally {
    a = 0;
}
callAnotherMethod();
```

编译之后是这样：

```
const/16 v0, 0xa

:try_start_0            # try 块开始
invoke-direct {p0}, Lnet/flygon/myapplication/SubActivity;->call
SomeMethod()V
:try_end_0              # try 块结束

.catchall {:try_start_0 .. :try_end_0} :catchall_0

const/4 v0, 0x0         # 复制一份到外面
invoke-direct {p0}, Lnet/flygon/myapplication/SubActivity;->call
AnotherMethod()V
return-void

:catchall_0             # finally 块开始
move-exception v1
const/4 v0, 0x0
throw v1                # finally 块结束
```

我们可以看到，编译器把 `finally` 编译成了重新抛出的 `.catch-all` ，这在逻辑上也是说得通的。但是， `finally` 中的逻辑在无异常情况下也会执行，所以需要复制一份到 `finally` 块的后面。

try-catch-finally

下面看看如果把这两个叠加起来会怎么样。

```
int a = 10;
try {
    callSomeMethod();
} catch (Exception e) {
    a = 1;
}
finally {
    a = 0;
}
callAnotherMethod();
```

```
const/16 v0, 0xa

:try_start_0            # try 块开始
invoke-direct {p0}, Lnet/flygon/myapplication/SubActivity;->call
SomeMethod()V
:try_end_0              # try 块结束

.catch Ljava/lang/Exception; {:try_start_0 .. :try_end_0} :catch
_0
.catchall {:try_start_0 .. :try_end_0} :catchall_0

const/4 v0, 0x0         # 复制一份到外面

:goto_0
invoke-direct {p0}, Lnet/flygon/myapplication/SubActivity;->call
AnotherMethod()V
return-void

:catch_0                # catch 块开始
move-exception v1
const/4 v0, 0x1
const/4 v0, 0x0         # 复制一份到 catch 块里面
goto :goto_0            # catch 块结束

:catchall_0             # finally 块开始
move-exception v2
const/4 v0, 0x0
throw v2                # finally 块结束
```

我们可以看到，其中同时含有 `.catch` 块和 `.catchall` 块。有一些不同之处在于， `finally` 块中的语句异常发生时也要执行，并且如果把 `finally` 编译成 `.catchall` ，那么和 `.catch` 就是互斥的，所以要复制一份到 `catch` 块里面。特别是 `finally` 块中的语句一多，就容易乱。

锁

指令	含义
monitor-enter vx	获得 `vx` 所引用的对象的锁
monitor-exit vx	释放 `vx` 所引用的对象的锁

对应 Java 的 `synchronized` 语句。而 `synchronized` 一般是被 `try-finally` 包起来的。

如果你编写：

```
int a = 1;
synchronized(this) {
    a = 2;
}
```

就相当于

```
int a = 1;
// monitor-enter this
try {
    a = 2;
} finally {
    // monitor-exit this
}
```

此外 Java 中没有与这两条指令相对应的方法，所以这两条指令一定成对出现。

数据转换

整数与浮点以及浮点与浮点

```
int-to-float vx, vy
int-to-double vx, vy
long-to-float vx, vy
long-to-double vx, vy
float-to-int vx, vy
float-to-long vx,vy
float-to-double vx, vy
double-to-int vx, vy
double-to-long vx, vy
double-to-float vx, vy
```

因为它们的表示方式不同，所以要保持表示的值不变，重新计算二进制位。如果不转换的话，就相当于二进制位不变，而表示的值改变，结果毫无意义。比如前面的 `0.1f` 如果不转换为直接使用，就会表示 `0x3dcccccd` 。

整数之间的向上转换

这种转换方式相当直接， `int` 向 `long` 转换， `long` 的第一个寄存器完全复制，第二个寄存器以 `int` 的最高位填充。除此之外没有其它的指令了，因为比 `int` 小的整数其实都是 32 位表示的，只是有效范围是 8 位或 16 位罢了（见数据定义）。

```
int-to-long vx,vy
```

整数之间的向下转换

其规则是数据位截断，符号位保留。每个整数的最高位都是符号位，其余是数据位。以 `int` 转 `short` 为例，`int` 的低 15 位复制给 `short` ，然后 `int` 的最高位（符号位）复制给 `short` 的最高位。其它同理。如果不转换而直接使用的话，会直接截断低 16 位，符号可能不能保留。

```
long-to-int vx,vy
int-to-byte vx,vy
int-to-char vx,vy
int-to-short vx,vy
```

NOP

`nop` 指令表示无操作。在一些场合下，不能修改二进制代码的字节数和偏移，需要用 `nop` 来填充，但是安卓逆向中几乎用不到。

参考

- Bytecode for the Dalvik VM
- Dalvik字节码含义查询表
- DVM 指令集图解

安卓逆向系列教程（二）APK 和 DEX

作者：飞龙

APK

APK 是 Android 软件包的分发格式，它本身是个 Zip 压缩包。APK 根目录下可能出现的目录和文件有：

名称	用途
`META-INF`	存放元数据
`AndroidManifest.xml`	编译后的全局配置文件
`assets`	存放资源文件，不会编译
`classes.dex`	编译并打包后的源代码
`lib`	存放二进制共享库，含有 `armeabi-*`、`mips`、`x86` 等文件夹，对应具体的平台
`res`	存放资源文件
`resources.arsc`	编译并打包后的 `res/values` 中的文件

`res`

res 中可能出现的目录如下：

名称	用途
`anim`	存放编译后的动画 XML 文件（ `<XXXAnimation>` ）
`color`	存放编译后的选择器 XML 文件（ `<selector>` ）
`drawable-*`	存放图片， `*` 为不同分辨率，图片按照不同分辨率归类。其中带 `.9` 的图片为可拉伸的图片。
`layout`	存放编译后的布局 XML 文件（ `<XXXLayout>` ）
`menu`	存放编译后的菜单 XML 文件（ `<menu>` ）
`mipmap-*`	存放使用 mipmap 技术加速的图片，一般用来存放应用图标，其它同 `drawable-*`
`raw`	存放资源文件，不会编译，比如音乐、视频、纯文本等
`xml`	存放编译后的自定义 XML 文件

`resources.arsc`

在 APK 中是找不到 `res/values` 这个目录的，因为它里面的文件编译后打包成了 `resources.arsc` 。为了理解它，我们先看一看原始的 `res/values` 。

`res/values` 中保存资源 XML 文件，根节点为 `<resources>` 。一般可能会出现以下几种文件：

名称	用途
`arrays.xml`	存放整数数组和字符串数组，使用 `<integer-array>` 或 `<string-array>` 定义，元素使用 `<item>` 定义
`bools.xml`	存放布尔值，使用 `<bool>` 定义
`colors.xml`	存放颜色，使用 `<color>` 定义
`dimens.xml`	存放尺寸，使用 `<dimen>` 定义
`drawables.xml`	存放颜色，使用 `<drawable>` 定义
`ids.xml`	存放 ID，使用 `<item type="id">` 定义
`integers.xml`	存放整数，使用 `<integers>` 定义
`strings.xml`	存放字符串，使用 `<strings>` 定义
`styles.xml`	存放颜色，使用 `<style>` 定义，元素使用 `<item>` 定义

`res/values` 中的文件名称是无所谓的，这些名称只是约定。也就是说，任何 `res/values` 中的文件中的字符串都会出现在 `R.strings` 里面。

虽然我们在 APK 中无法直接看到这些文件，但是反编译之后就可以了。反编译之后，我们也会找到一个 `public.xml` 文件，是 `res` 里所有东西的索引：

```
<resources>
    <public type="drawable" name="ic_launcher" id="0x7f020000" /
>
    <public type="layout" name="activity_main" id="0x7f030000" /
>
    <public type="layout" name="activity_sub" id="0x7f030001" />
    <public type="dimen" name="activity_horizontal_margin" id="0
x7f040000" />
    <public type="dimen" name="activity_vertical_margin" id="0x7
f040001" />
    <public type="string" name="action_settings" id="0x7f050000"
 />
    <public type="string" name="app_name" id="0x7f050001" />
    <public type="string" name="hello_world" id="0x7f050002" />
    <public type="string" name="title_activity_sub" id="0x7f0500
03" />
    <public type="style" name="AppTheme" id="0x7f060000" />
    <public type="menu" name="main" id="0x7f070000" />
    <public type="menu" name="sub" id="0x7f070001" />
    <public type="id" name="button1" id="0x7f080000" />
    <public type="id" name="action_settings" id="0x7f080001" />
</resources>
```

DEX

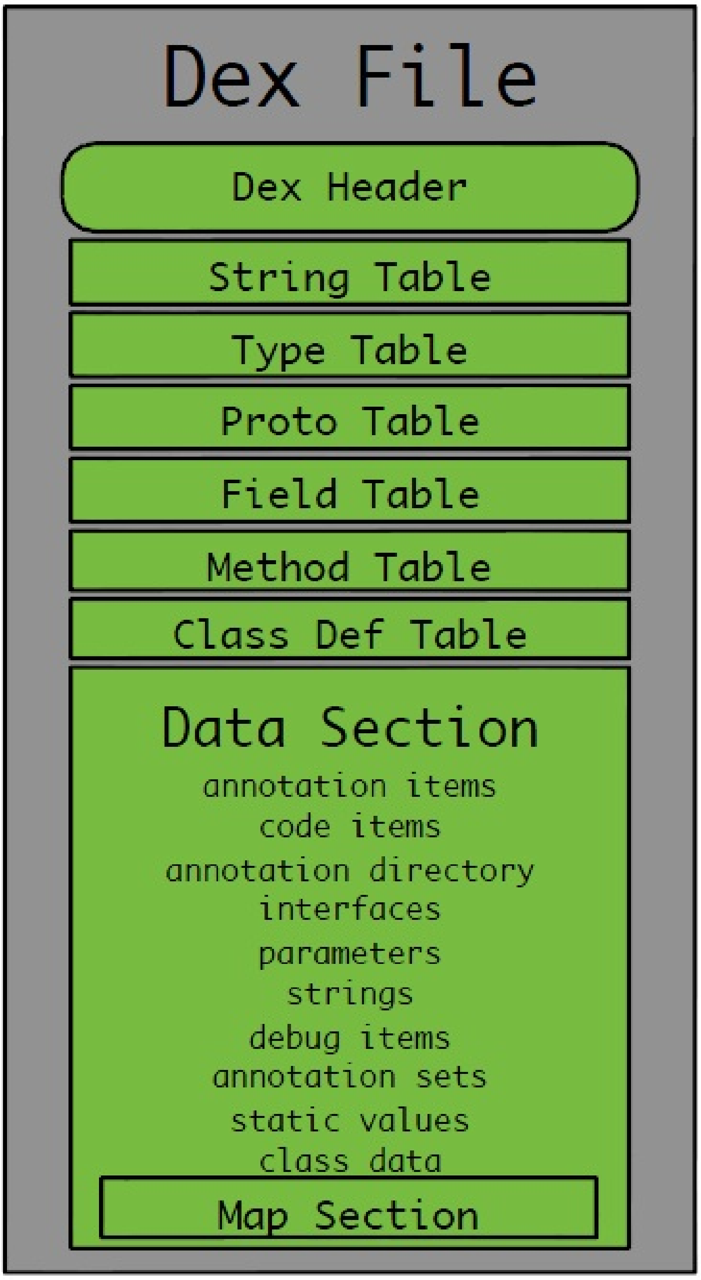
DEX 即 Dalvik Executable，Dalvik 可执行文件。它的结构如下：

```
struct DexFile{
    DexHeader     Header;
    DexStringId   StringIds[stringIdsSize];
    DexTypeId     TypeIds[typeIdsSize];
    DexFieldId    FieldIds[fieldIdsSize];
    DexMethodId   MethodIds[methodIdsSize];
    DexProtoId    ProtoIds[protoIdsSize];
    DexClassDef   ClassDefs[classDefsSize];
    DexData       Data;
    DexLink       LinkData;
};
```

我们可以看到，它可以分为九个区段，如下：

Header
StringIds
TypeIds
FieldIds
MethodIds
ProtoIds
ClassDefs
Data
LinkData

大体结构如这张图所示：



另外，在讲解各个区段之前，需要首先了解一些数据类型的定义：

类型	定义
`u1`	等同于 `uint8_t` ，表示 1 字节的无符号数
u2	等同于 uint16_t ，表示 2 字节的无符号数
`u4`	等同于 `uint32_t` ，表示 4 字节的无符号数
u8	等同于 uint64_t ，表示 8 字节的无符号数

Header 区段

Header 区段用于储存版本标识、校验和、文件大小、各部分的大小及偏移。结构以及描述如下：

```
struct DexHeader {
    u1  magic[8];           /* 版本标识 */
    u4  checksum;           /* adler32 检验和 */
    u1  signature[kSHA1DigestLen]; /* SHA-1 哈希值 */
    u4  fileSize;           /* 整个文件大小 */
    u4  headerSize;         /* Header 区段大小 */
    u4  endianTag;          /* 字节序标记 */
    u4  linkSize;           /* 链接区段大小 */
    u4  linkOff;            /* 链接区段偏移 */
    u4  mapOff;             /* MapList 的偏移 */
    u4  stringIdsSize;      /* StringId 的个数 */
    u4  stringIdsOff;       /* StringIds 区段偏移 */
    u4  typeIdsSize;        /* TypeId 的个数 */
    u4  typeIdsOff;         /* TypeIds 区段偏移 */
    u4  protoIdsSize;       /* ProtoId 的个数 */
    u4  protoIdsOff;        /* ProtoIds 区段偏移 */
    u4  fieldIdsSize;       /* FieldId 的个数 */
    u4  fieldIdsOff;        /* FieldIds 区段偏移 */
    u4  methodIdsSize;      /* MethodId 的个数 */
    u4  methodIdsOff;       /* MethodIds 区段偏移 */
    u4  classDefsSize;      /* ClassDef 的个数 */
    u4  classDefsOff;       /* ClassDefs 区段偏移 */
    u4  dataSize;           /* 数据区段的大小 */
    u4  dataOff;            /* 数据区段的文件偏移 */
};
```

有几个条目需要特别提醒。

- `magic`：必须为 `DEX_FILE_MAGIC`：

```
ubyte[8] DEX_FILE_MAGIC = { 0x64 0x65 0x78 0x0a 0x30 0x33 0x35 0x00 }
                        = "dex\n035\0";
```

- `checksum`：是整个文件除去它本身以及魔数的校验和。
- `signature`：是整个文件除去它本身、校验和以及魔数的哈希值。
- `headerSize`：一般为 70。
- `endianTag`：有两种顺序，小端和大端，定义如下：

```
uint ENDIAN_CONSTANT = 0x12345678; /* 小端序 */
uint REVERSE_ENDIAN_CONSTANT = 0x78563412; /* 大端序 */
```

一般为小端序，反正我还没见过大端的。

- `stringIdsOff` ：由于前一个区段的偏移加上它的长度一般为后一个区段的偏移，所以这个条目一般也为 70。
- `xxxSize` ：要注意有几个是个数，后缀也是`Size` 。
- `xxxOff` ：如果对应的`xxxSize` 为 0，那么它也为 0（很奇怪）。

StringIds 区段

StringIds 区段包含`stringIdsSize` 个`DexStringId` 结构，如下：

```
struct DexStringId {
    u4 stringDataOff;    /* 字符串内容，字符串数据偏移 */
};
```

其中数据偏移指向 Data 区段的字符串数据。

TypeIds 区段

TypeIds 包含`typeIdsSize` 个`DexTypeId` 结构，如下：

```
struct DexTypeId {
    u4 descriptorIdx;     /* 类型的完全限定符，指向 DexStringId 列表的索引 */
};
```

索引是一个从 0 开始的数字，表示对应第几个`DexStringId` 。这些`DexStringId` 指向的字符串都是类型名称，比如`I` 、`Ljava/lang/String;` 之类的。`DexTypeId` 的索引也会用于后面的结构。

ProtoIds 区段

ProtoIds 包含`ProtoIdsSize` 个`DexProtoId` 结构。这里的 Proto 指方法原型，包含返回类型和参数类型。

```
struct DexProtoId {
    u4 shortyIdx;       /* 原型缩写，指向 DexStringId 列表的索引 */
    u4 returnTypeIdx;   /* 返回类型，指向 DexTypeId 列表的索引 */
    u4 parametersOff;   /* 参数类型列表，指向 DexTypeList 的偏移 */
};

struct DexTypeList {
    u4 size;             /* 接下来 DexTypeItem 的个数 */
    DexTypeItem list[size]; /* DexTypeItem 结构 */
};

struct DexTypeItem {
    u2 typeIdx;     /* 参数类型，指向 DexTypeId 列表的索引 */
};
```

原型缩写是把所有返回类型和参数类型的名称拼在一起，对象的话只写 `L` 。比如 `int(int,int)` 写为 `III` ， `void()` 写为 `V` ， `void(String)` 写为 `VL` 。

参数类型列表一般保存在 `Data` 区段中，如果没有， `parametersOff` 为 0。

FieldIds 区段

TypeIds 包含 `fieldIdsSize` 个 `DexFieldId` 结构，如下：

```
struct DexFieldId {
    u2 classIdx;    /* 类的类型，指向 DexTypeId 列表的索引 */
    u2 typeIdx;     /* 字段类型，指向 DexTypeId 列表的索引 */
    u4 nameIdx;     /* 字段名称，指向 DexStringId 列表的索引 */
};
```

MethodIds 区段

MethodIds 包含 `methodIdsSize` 个 `DexMethodId` 结构，如下：

```
struct DexMethodId {
    u2 classIdx;   /* 类的类型，指向 DexTypeId 列表的索引 */
    u2 protoIdx;   /* 方法原型，指向 DexProtoId 列表的索引 */
    u4 nameIdx;    /* 方法名称，指向 DexStringId 列表的索引 */
};
```

ClassDefs 区段

ClassDefs 包含 `classDefsSize` 个 `DexClassDef` 结构，如下：

```
struct DexClassDef {
    u4 classIdx;         /* 类的类型，指向 DexTypeId 列表的索引 */
    u4 accessFlags;      /* 访问标志 */
    u4 superclassIdx;    /* 父类类型，指向 DexTypeId列表的索引 */
    u4 interfacesOff;    /* 接口，指向 DexTypeList 的偏移 */
    u4 sourceFileIdx;    /* 源文件名，指向 DexStringId 列表的索引 */
    u4 annotationsOff;   /* 注解，指向 DexAnnotationsDirectoryItem
 结构 */
    u4 classDataOff;     /* 指向 DexClassData 结构的偏移 */
    u4 staticValuesOff;  /* 指向 DexEncodedArray 结构的偏移 */
};

struct DexClassData {
    DexClassDataHeader header;                       /* 各个字段与方
法的个数 */
    DexField staticFields[staticFieldsSize];         /* 静态字段 */
    DexField instanceFields[instanceFieldsSize];     /* 实例字段 */
    DexMethod directMethods[directMethodsSize];      /* 直接方法 */
    DexMethod virtualMethods[virtualMethodsSize];    /* 虚方法 */
};

struct DexClassDataHeader {
    u4 staticFieldsSize;   /* 静态字段个数 */
    u4 instanceFieldsSize; /* 实例字段个数 */
    u4 directMethodsSize;  /* 直接方法个数 */
    u4 virtualMethodsSize; /* 虚方法个数 */
};

struct DexField {
    u4 fieldIdx;    /* 指向 DexFieldId 的索引 */
    u4 accessFlags; /* 访问标志 */
};

struct DexMethod {
    u4 methodIdx;   /* 指向 DexMethodId 的索引 */
    u4 accessFlags; /* 访问标志 */
    u4 codeOff;     /* 方法指令，指向DexCode结构的偏移 */
};

struct DexCode {
    u2 registersSize;    /* 使用的寄存器个数 */
    u2 insSize;          /* 参数个数 */
    u2 outsSize;         /* 调用其他方法时使用的寄存器个数 */
    u2 triesSize;        /* Try/Catch个数 */
    u4 debugInfoOff;     /* 指向调试信息的偏移 */
    u4 insnsSize;        /* 指令集个数，以2字节为单位 */
    u2 insns[insnsSize]; /* 指令集 */
};
```

`DexClassData` 和 `DexCode` 保存在 Data 区段中。

Data 区段

这个区段中除了存放二级结构和字符串，还有个重要的结构叫做 `DexMapList` ，它实际上 DEX 中所有东西的索引，包括各种二级结构、字符串和它本身。DEX 中同类结构都会保存在一起，所以一类结构只占用一个条目。

```
struct DexMapList {
    u4 size;                /* 条目个数 */
    DexMapItem list[size];  /* 条目列表 */
};

struct DexMapItem {
    u2 type;        /* 结构类型，kDexType 开头 */
    u2 unused;      /* 未使用，用于字节对齐 */
    u4 size;        /* 连续存放的结构个数 */
    u4 offset;      /* 结构的偏移 */
};

/* 结构类型代码 */
enum {
    kDexTypeHeaderItem               = 0x0000,
    kDexTypeStringIdItem             = 0x0001,
    kDexTypeTypeIdItem               = 0x0002,
    kDexTypeProtoIdItem              = 0x0003,
    kDexTypeFieldIdItem              = 0x0004,
    kDexTypeMethodIdItem             = 0x0005,
    kDexTypeClassDefItem             = 0x0006,
    kDexTypeMapList                  = 0x1000,
    kDexTypeTypeList                 = 0x1001,
    kDexTypeAnnotationSetRefList     = 0x1002,
    kDexTypeAnnotationSetItem        = 0x1003,
    kDexTypeClassDataItem            = 0x2000,
    kDexTypeCodeItem                 = 0x2001,
    kDexTypeStringDataItem           = 0x2002,
    kDexTypeDebugInfoItem            = 0x2003,
    kDexTypeAnnotationItem           = 0x2004,
    kDexTypeEncodedArrayItem         = 0x2005,
    kDexTypeAnnotationsDirectoryItem = 0x2006,
};
```

参考

- Android Dex文件结构解析
- Dalvik Executable Format
- DEX 结构图解

安卓逆向系列教程（三）工具篇

3.1 静态分析工具

> 作者：飞龙

以下工具可能都需要先安装 JDK，安装方法就不说了，随便一搜就是。

Android Killer

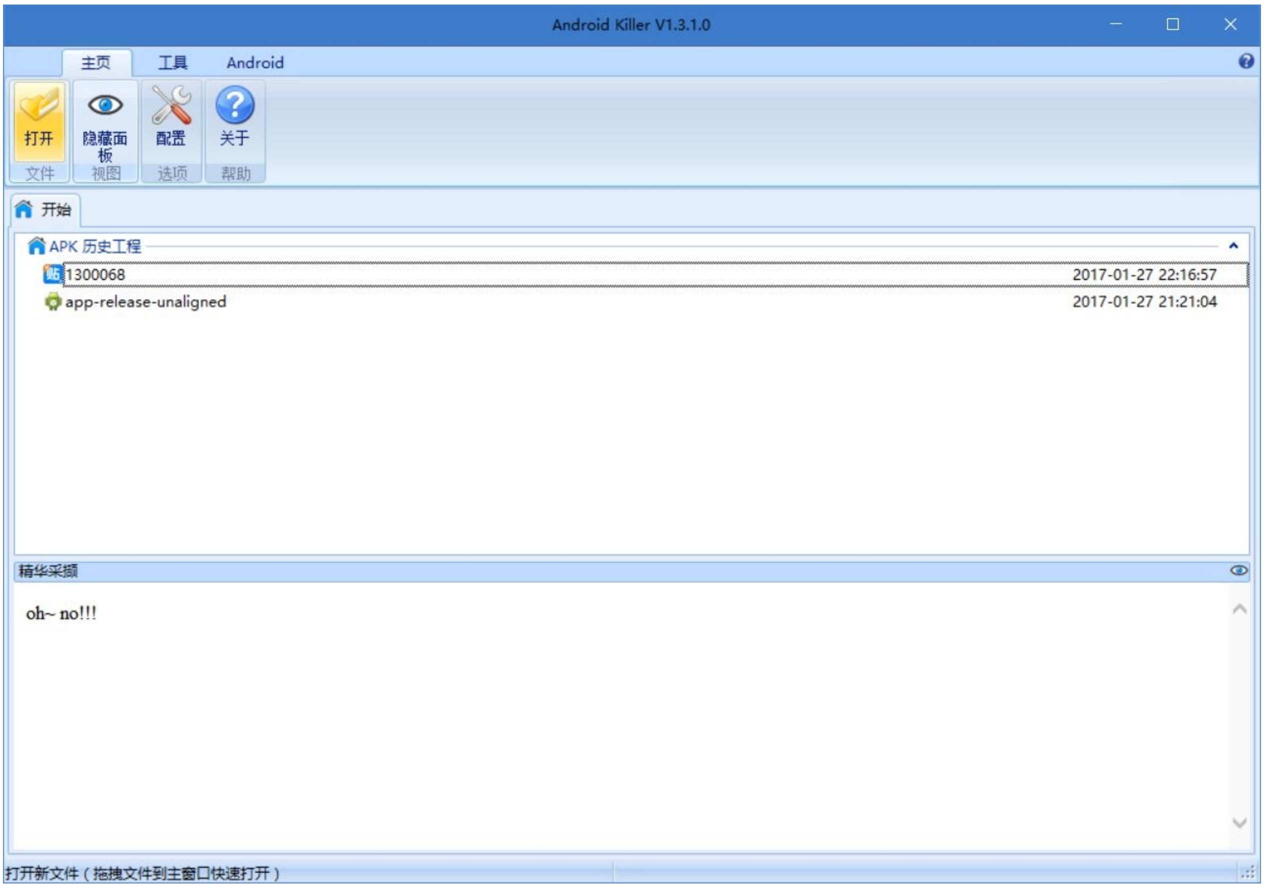
几年之前，我们要破解 APK，可能需要用到 apktool、dex2jar、jd-gui 以及 smali2java 等工具。还需要在控制台中键入命令，但现在有了集成工具，一切都变得省事了。

我们从这里下载 Android Killer。

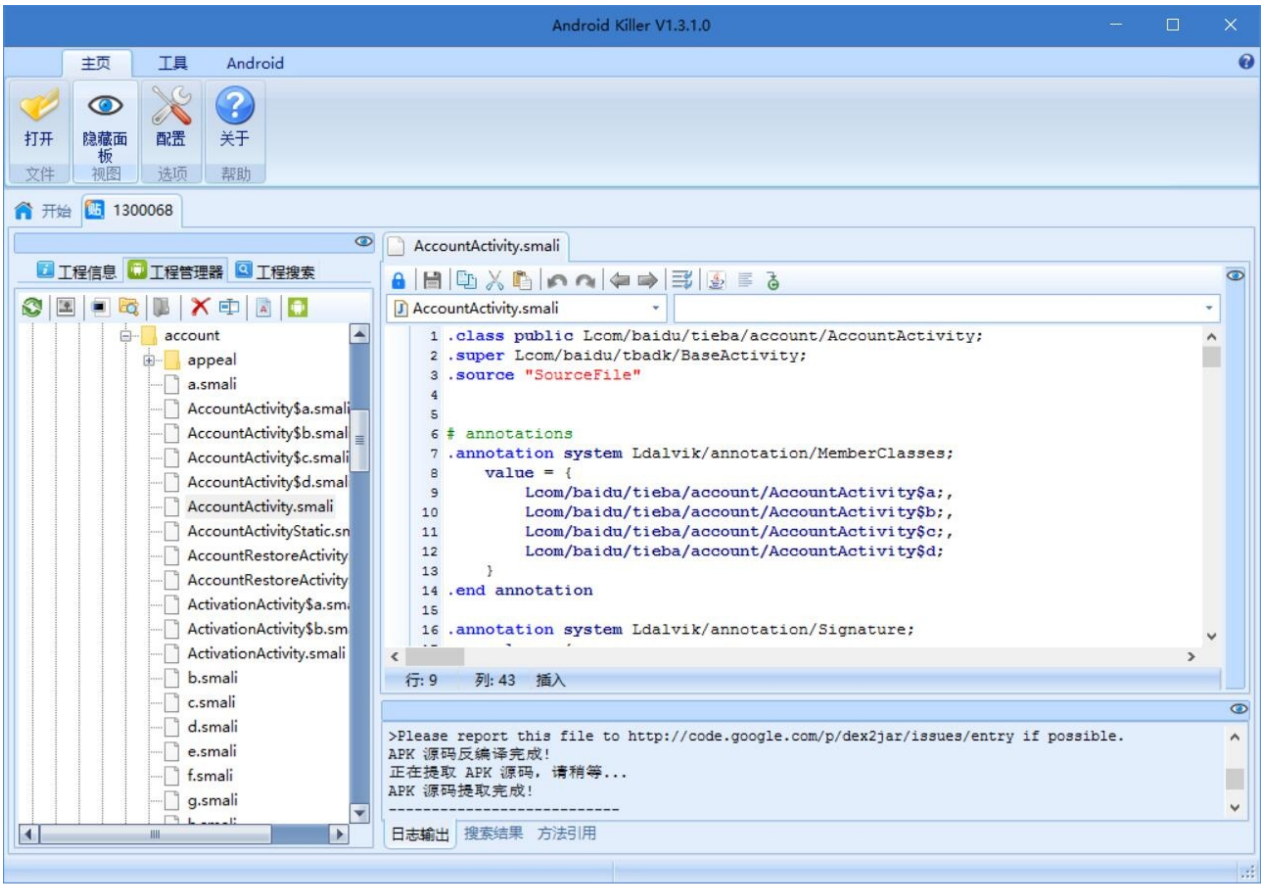
我们打开 `AndroidKiller.exe` ，它的启动界面是这样，很酷吧。



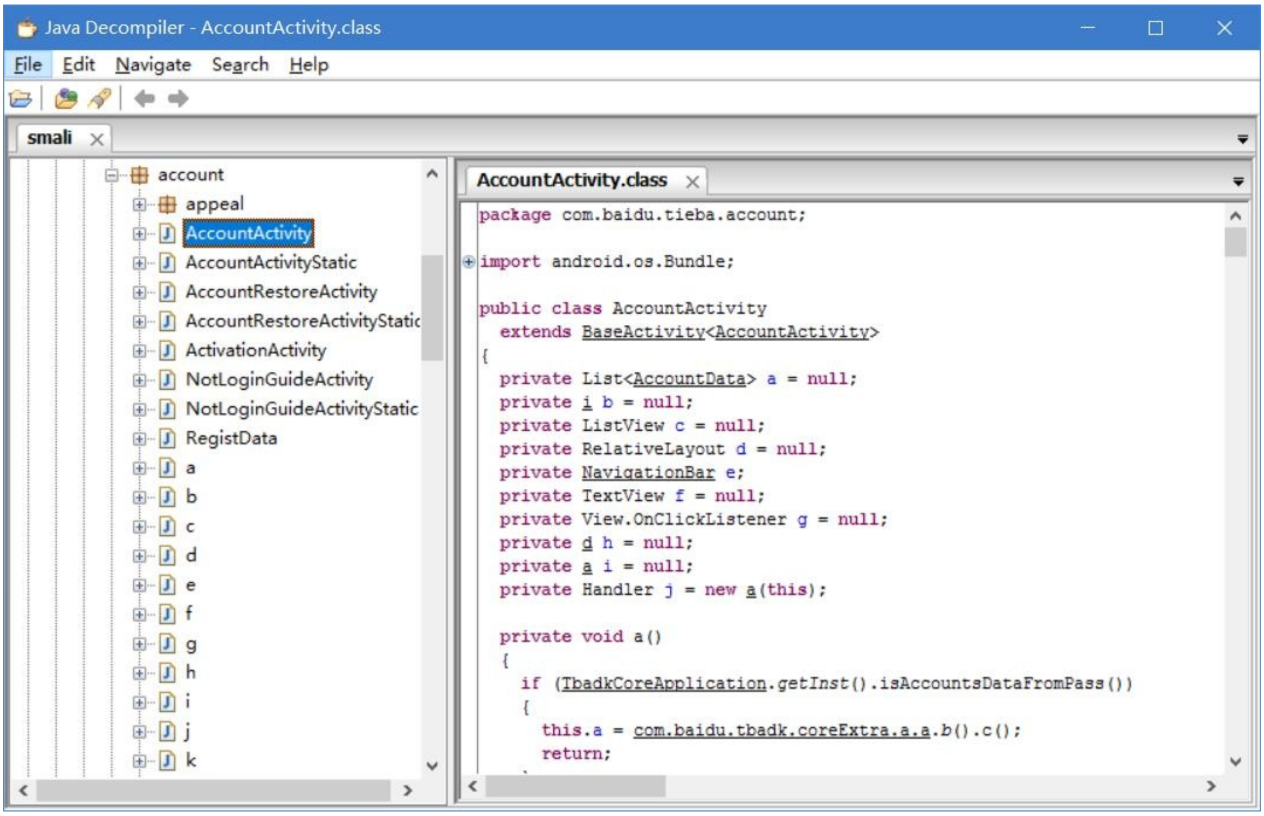
打开之后，点击左上角的“打开”按钮，选择要反编译的 APK，或者直接把 APK 拖进来。软件会马上开始反编译。



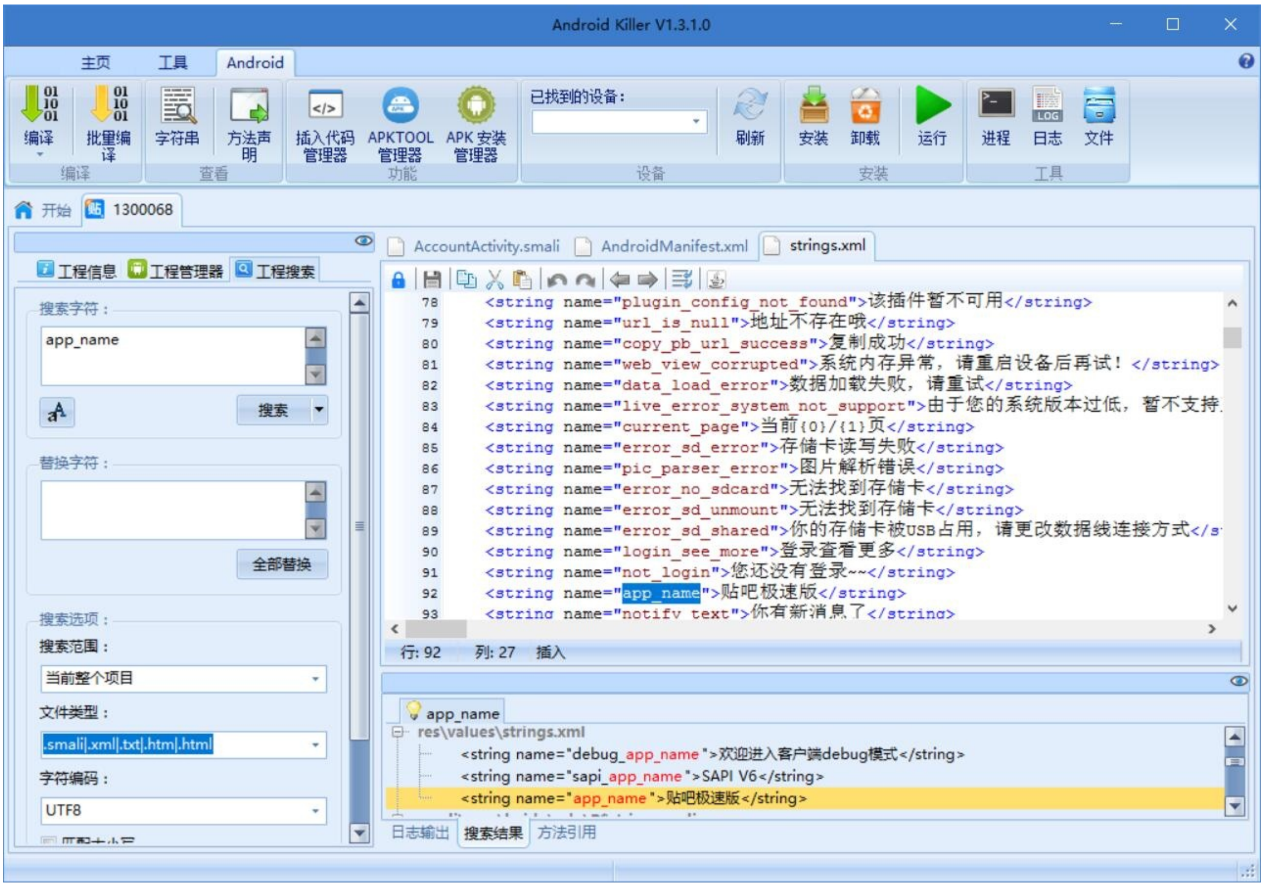
等一会儿，我们会看到反编译完成。之后切换到“工程管理器”，可以看到项目的结构，点击其中的文件可以在右边看到文件内容：



点击编辑框上方的 Java 图标，就会打开熟悉的 jd-gui 窗口：



我们切换到“工程搜索”，在下方的“搜索字符”输入框中输入 `app_name` ，点击下方的“搜索”。下方的框中会显示结果。我们点击结果，编辑框中会定位到具体文件。我们可以修改一下。



之后我们点击 `Android` 选项卡，点击第一项“编译”。



等一小会儿，重编译就完成了。



如果我们启动了模拟器，可以使用右边的几个按钮安装并运行。

此外，“工具”选项卡中有很多实用工具，大家可以一一尝试。

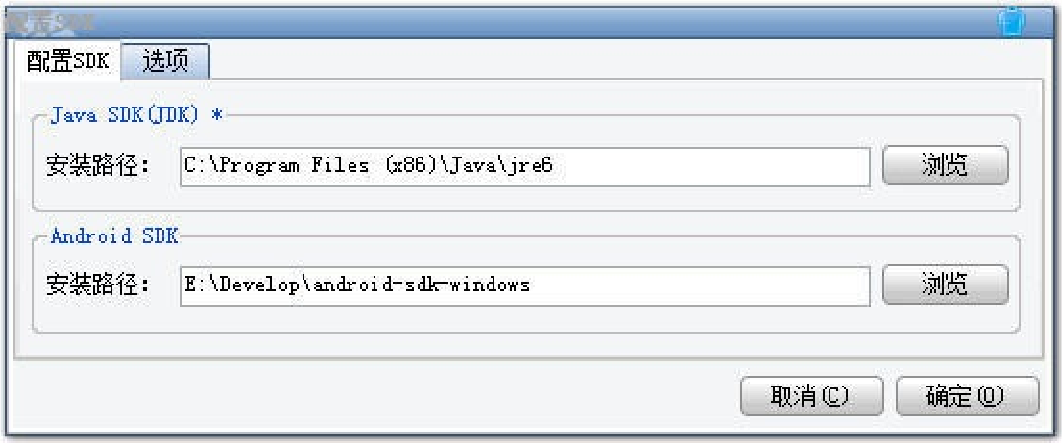


APK 改之理

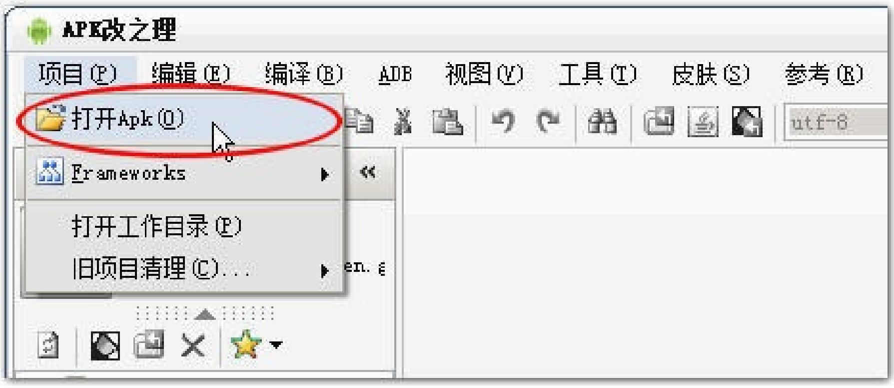
在这里下载软件。

双击 `ApkIDE.exe` 启动程序。如果是 XP 系统启动不了它，请下载安装 .Net Framework 2.0。

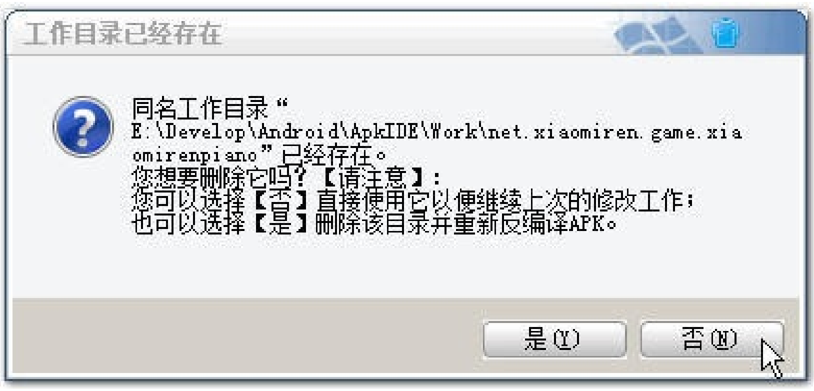
第一次启动时，软件会自动查找系统中的 JRE 安装目录，如果没有找到会提示你配置 SDK，可以点击菜单 “工具->配置SDK` 对 JDK 进行配置，如下图。JDK 的安装路径必须配置（如果不配置，则无法进行修改操作），Android SDK 则随意（有些功能需要用到它，比如 ddms 等，但这些功能都无关修改工作）。



单击菜单"项目->打开Apk"选择要修改的 Apk 文件（注：文件名称必须只有字母、数字、下划线、空格、点号等组成，不能包含中文或其它亚洲字符）。

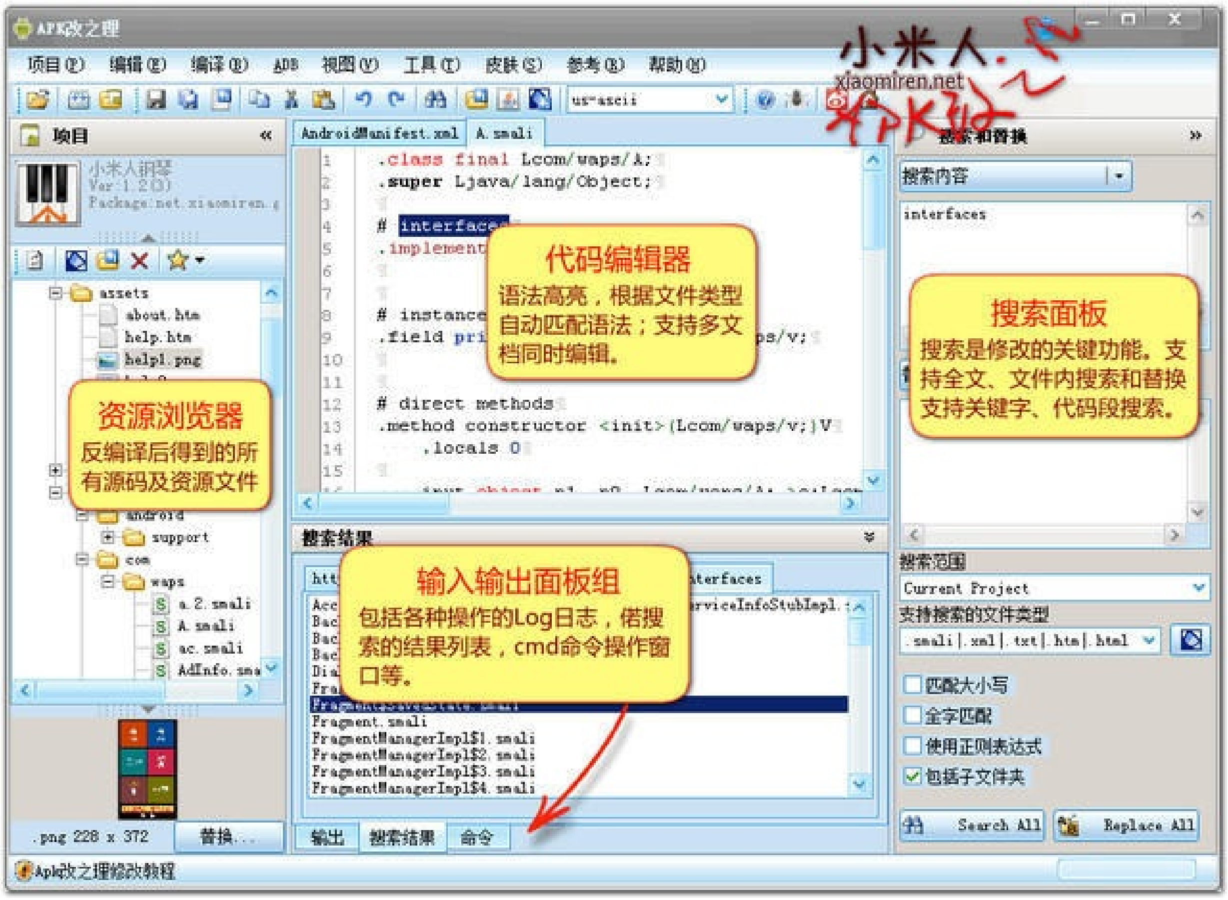


在打开 Apk 文件时 Apk 改之理会先对其进行基本的解析（包括它的名称、包、权限等)，然后根据该 apk 应用的包名生成它的同名工作目录，如果这个工作目录已经存在，Apk 改之理会询问是否要重新反编译 Apk。这里要注意，已有的工作目录通常是你以前修改这个 Apk 应用时所生成的工作目录，如果你要继续这个修改操作，则单击"否"继续使用它，否则就重新反编译得到一个全新的源代码。



提示：如果你想继续旧工作但却误点了"是"按钮，也不用担心，删除的目录被扔进了系统垃圾箱，你可以直接去系统回收站恢复。恢复时注意，如果你之前成功对这个应用进行过 dex2jar 操作（由软件在反编译 apk 时自动进行，但可能会因一些原因而失败），那么回收站中会看到两个同名的目录，选中它们右键恢复即可。（注：这个特性 Apk 改之理 2.1 或更高版本中有效）

现在你可以使用软件的搜索、替换等功能来对源代码进行修改，这种修改包括汉化、去广告、改名、替换资源、图片、xx 等等。下图中各个图标按钮都有提示文字，可以将鼠标悬浮在按钮上显示文字提示。具体的各项说明会单独写个文章来详细解释，基本上也没什么难点。



这里先提示一些没有说明的小功能：

（1）在文件树上，或搜索后得到的文件列表上，按住 Shift 键并单击鼠标右键会直接显示操作系统菜单。

（2）在“输入输出面板组”的搜索结果面板中，搜索结果列表以标签的形式各自分开，鼠标悬浮在标签上会显示对应搜索结果的搜索条件。

（3）工作目录下的第一个 `build` 目录下的文件不会被搜索(因为这个是 Apktool 编译时用到的，与我们的修改无直接关系)。

修改完成后单击菜单“编译->编译生成Apk”重新将源代码打包成 apk 文件，新生成的 apk 存放在原 apk 的同级目录下，其名称以 `ApkIDE_` 开头。

单击菜单“编译->获取生成的”可以直接在资源浏览器中定位到 apk 所在的目录。

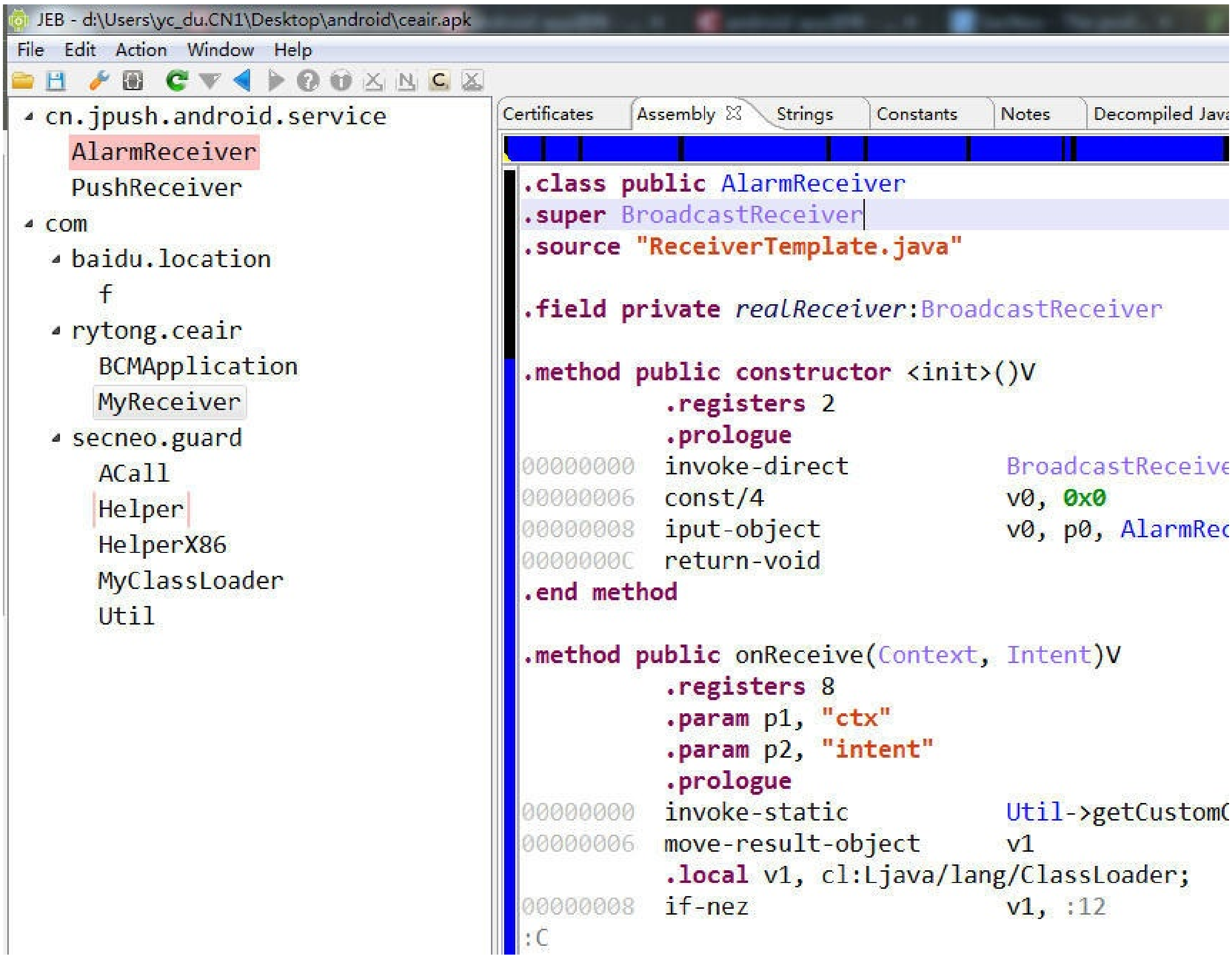
直接测试 Apk 需要用到菜单“ADB”下的菜单命令，如果你已经将设备连接到电脑，或者直接在电脑上打开了安卓模拟器，可以单击菜单“ADB->安装生成的APK”直接向设备或模拟器安装修改生成的 apk，然后再可以使用 `adb logcat` 来观察其运行状况。

如果发现 ADB 相关命令不起作用，你可以先用 `adb devices` 命令查看设备是否连接成功（可以直接在输入输出面板组的命令窗口输入 `adb devices` ），也可以使用菜单“工具->Dalvik Debug Monitor Service”（ddms）来测试。

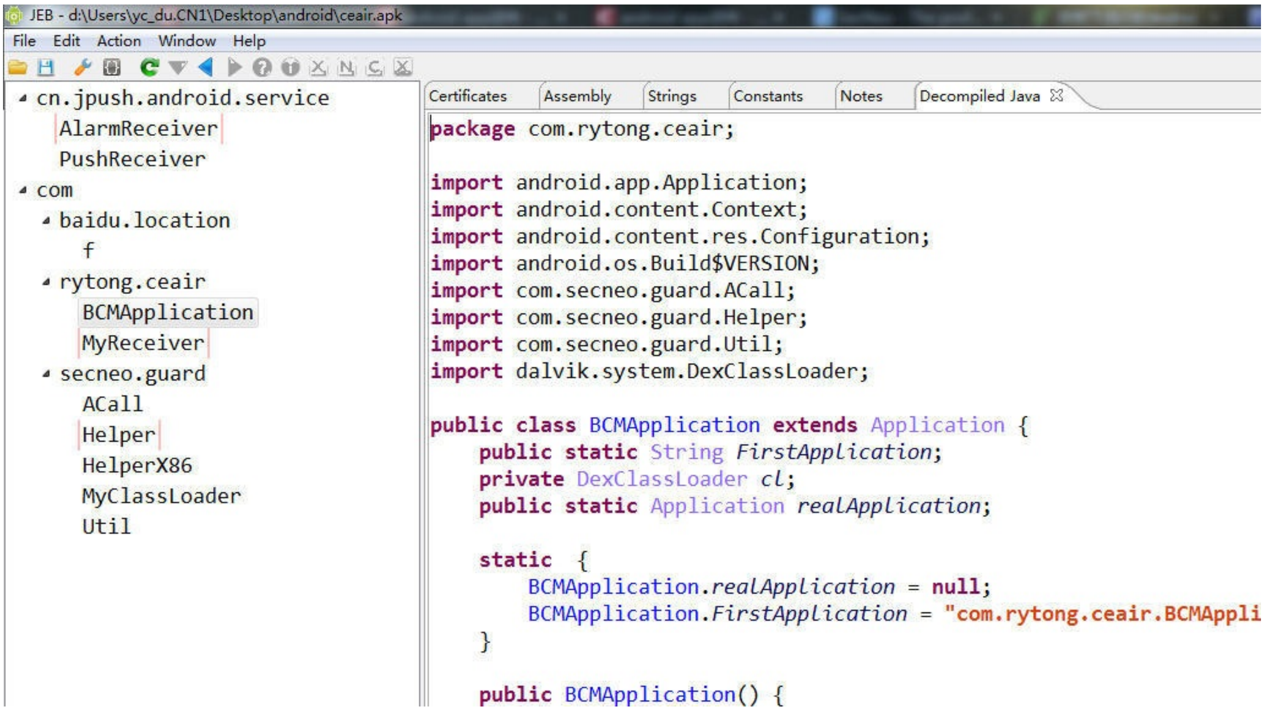
JEB

首先在这里下载软件。

打开软件之后，点击左上角的文件夹图标，之后选择要反编译的 APK 来打开文件。之后会进行反编译，完成后，主界面是这样：

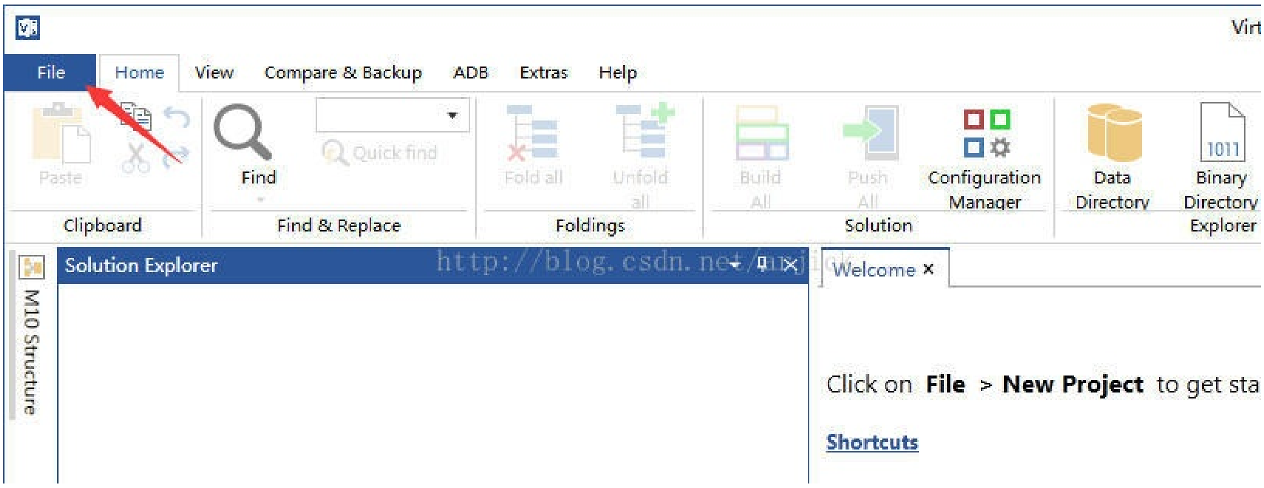


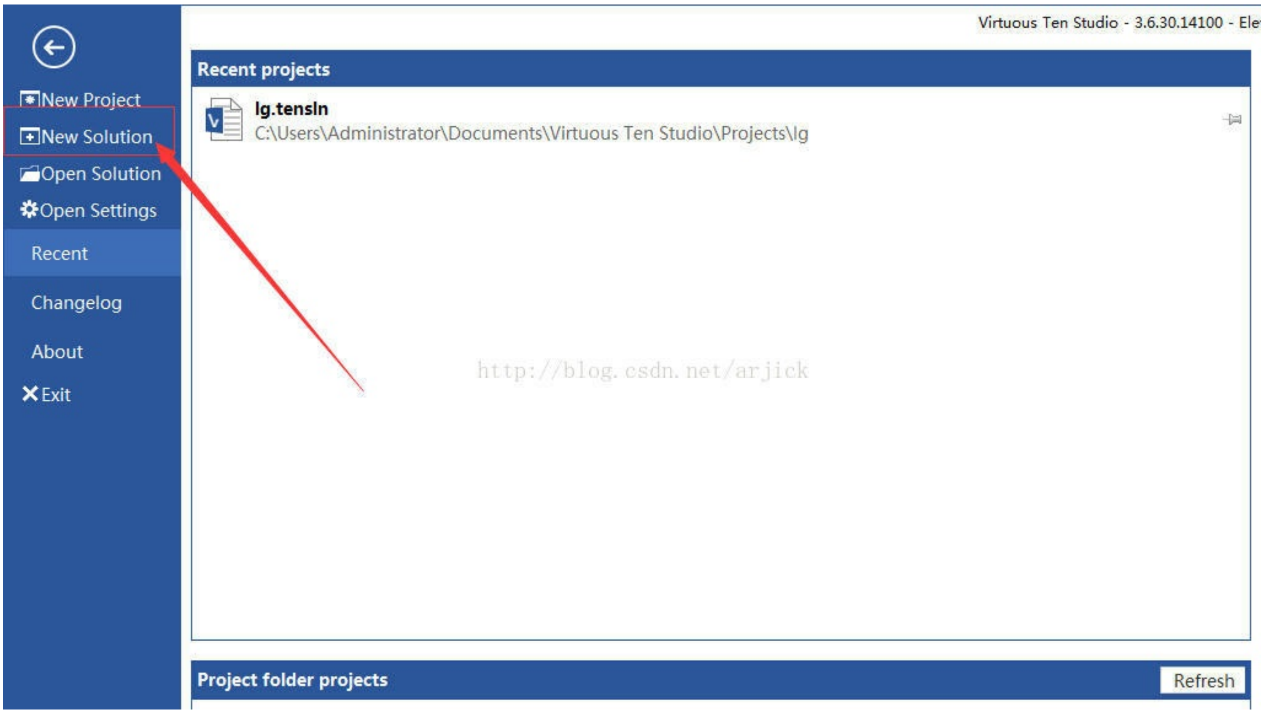
左边的树形图会显示项目的所有包和类。右边的编辑框中会显示 Smali 代码，以及字符串等资源。选择 `Decompiled Java` 选项卡，还会看到对应的 Java 代码。



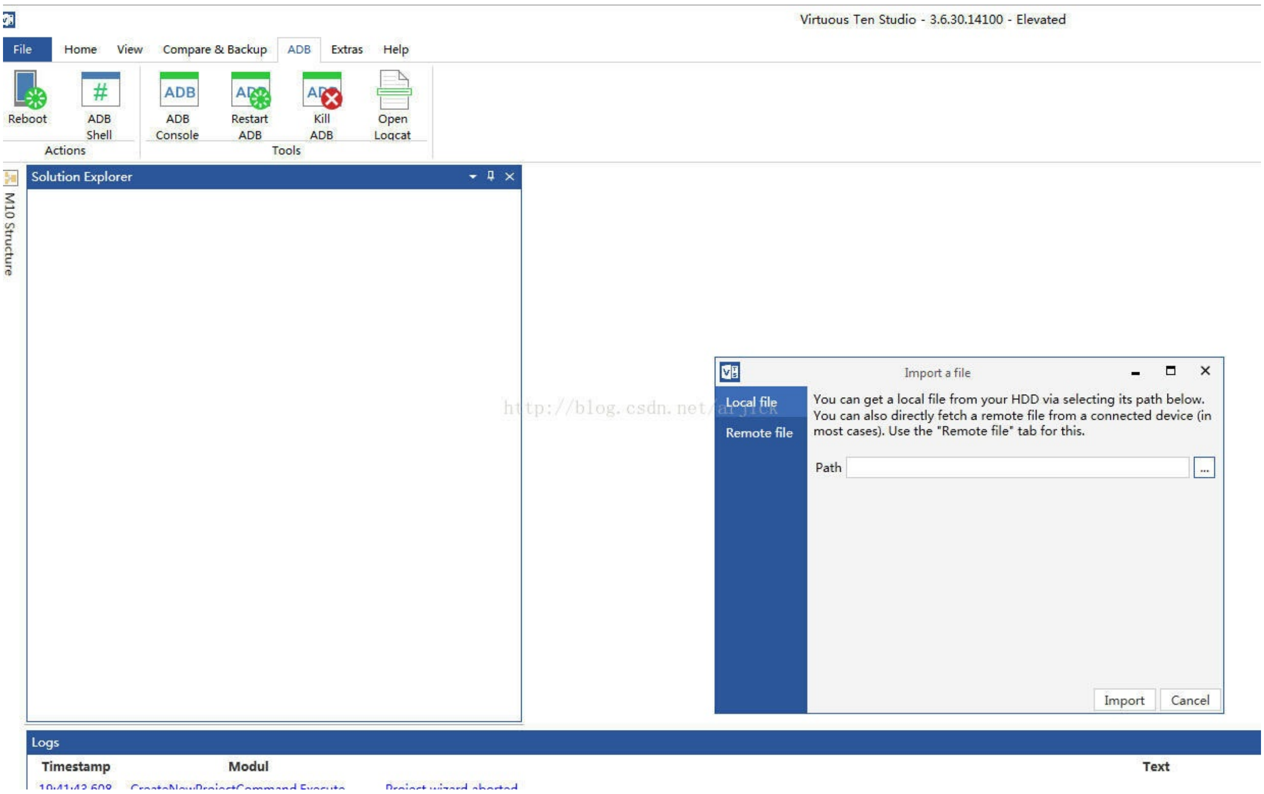
VTS（Virtuous Ten Studio）

打开 VTS 之后，首先我们需要点击 `File->New Solution` 新建一个 solution：

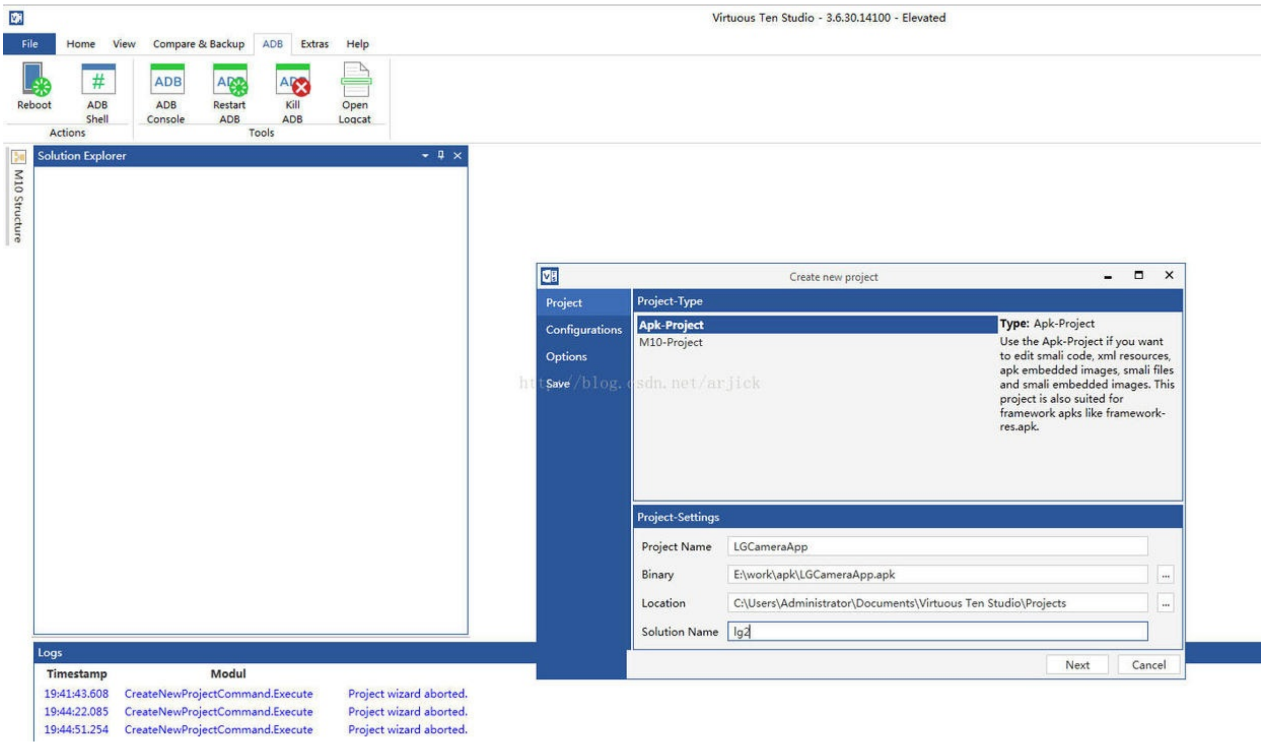




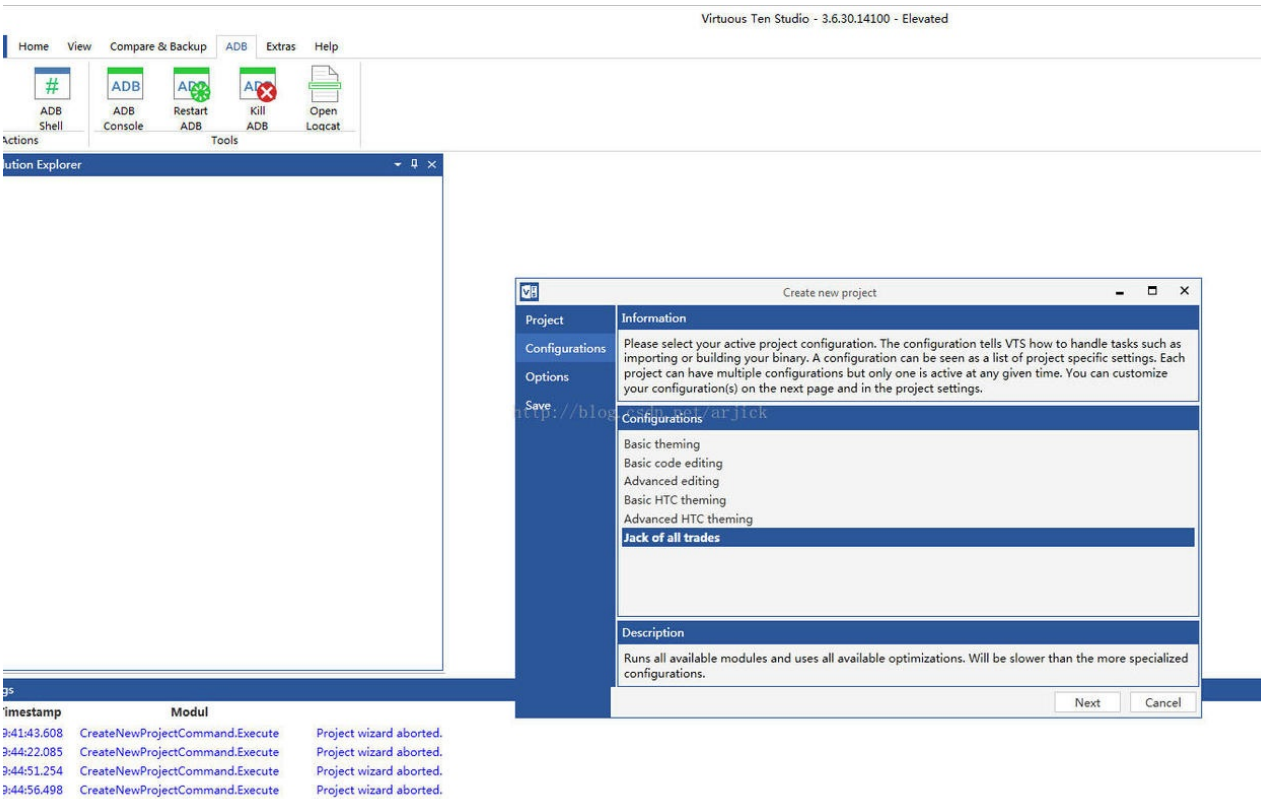
接下来需要选择要反编译的文件：



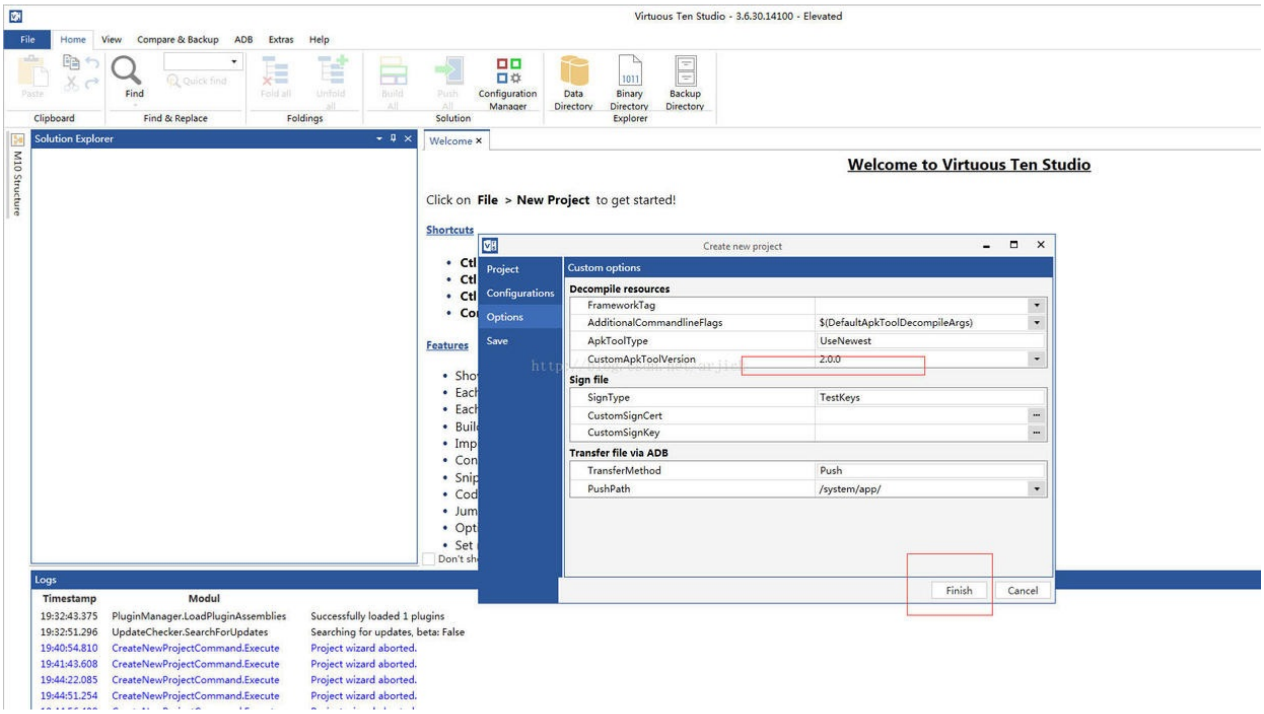
之后是项目类型、项目名称、解决方案名称及位置：



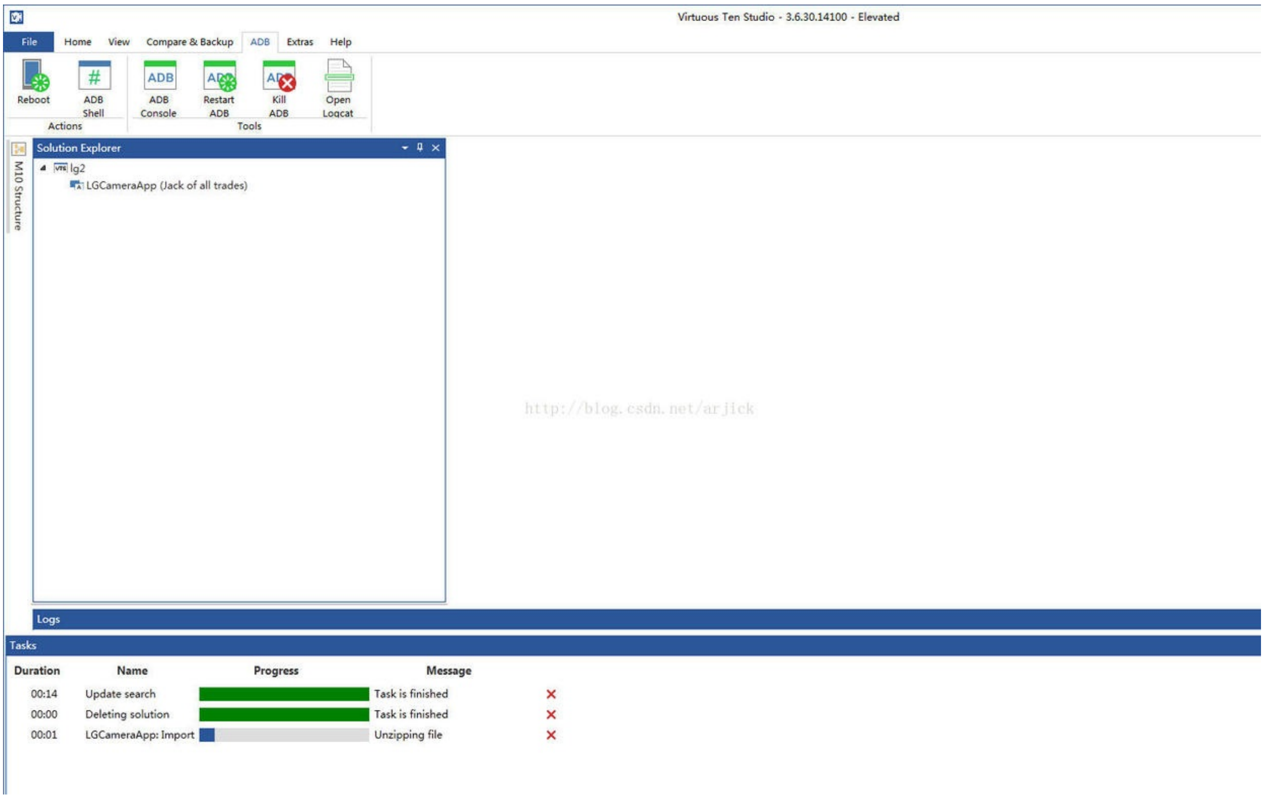
这里我们全选：



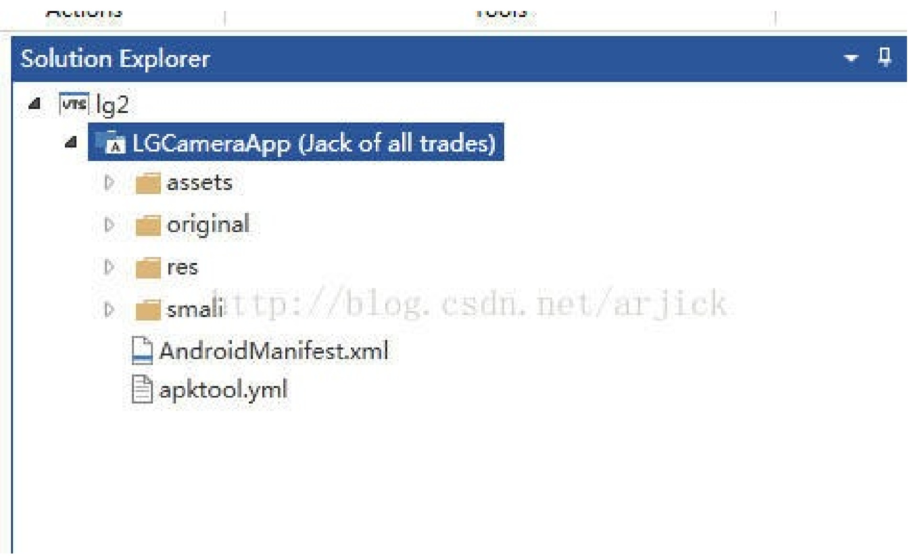
最后选择 Apktool 的版本：



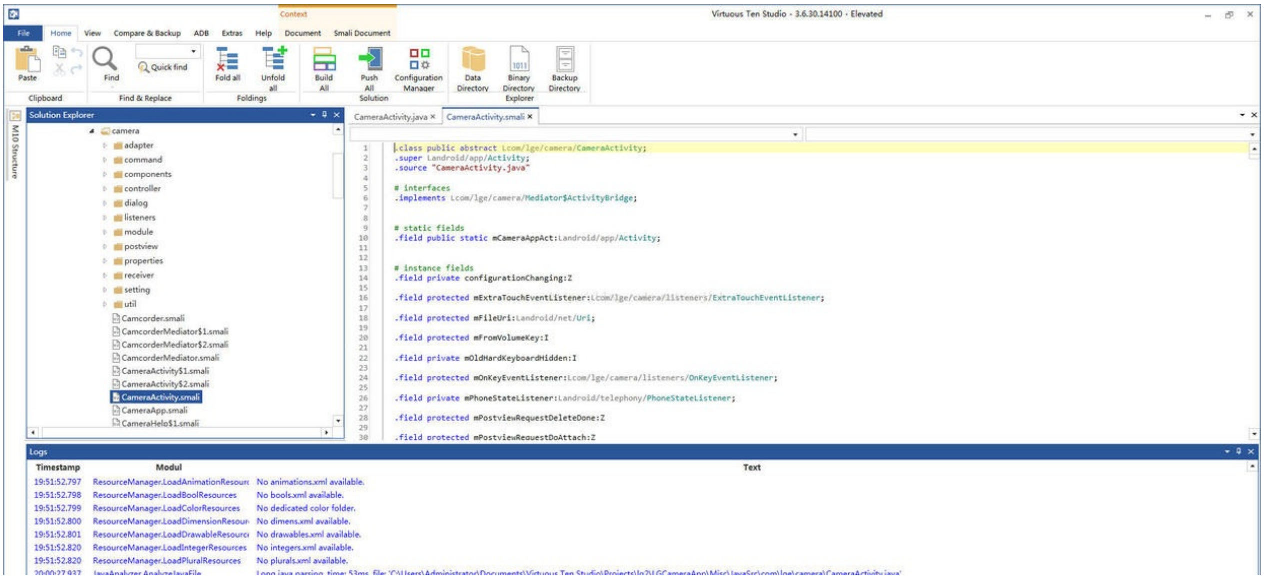
然后它会开始反编译：



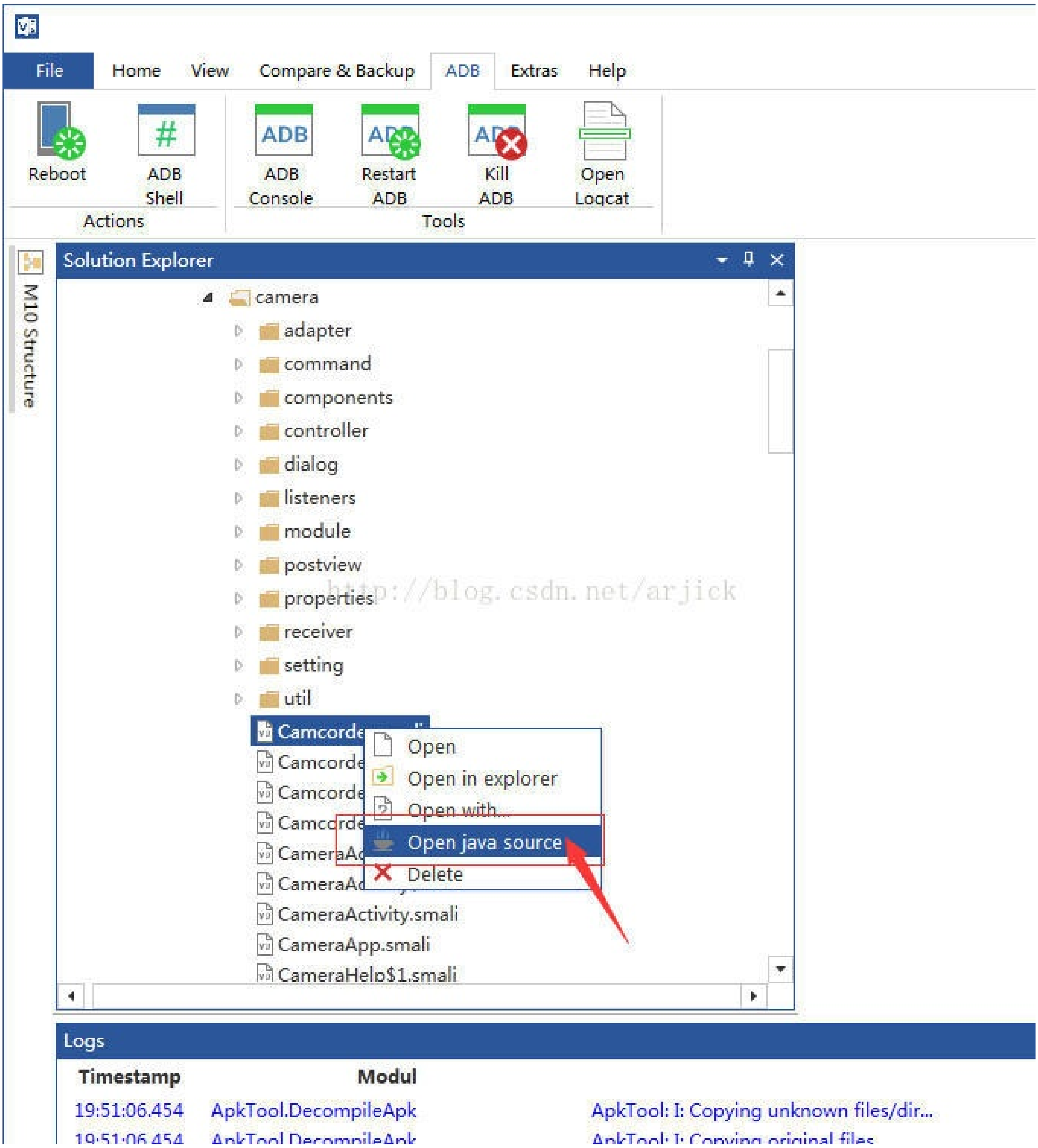
完成后可以在左侧看到目录：



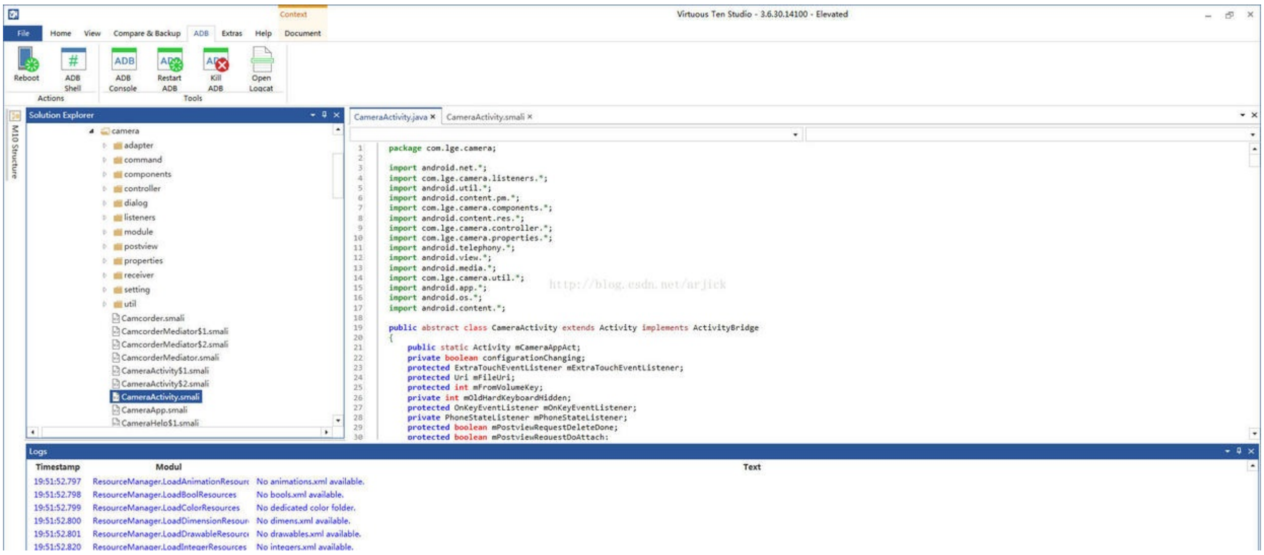
点击里面的文件可以查看 Smali 代码：



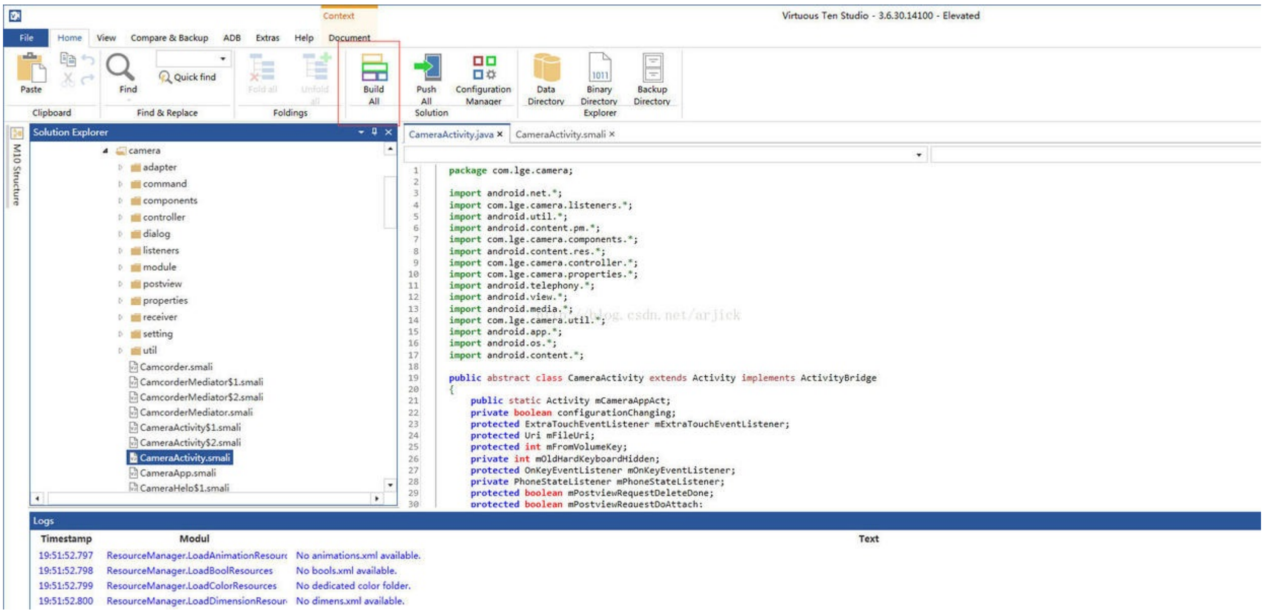
在文件上点击右键，会看到 `Open Java Source` ：



我们点击它，可以查看 Java 代码：



我们可以点击 `Home->Build All` 来重编译。



3.2 抓取手机封包

> 作者：飞龙

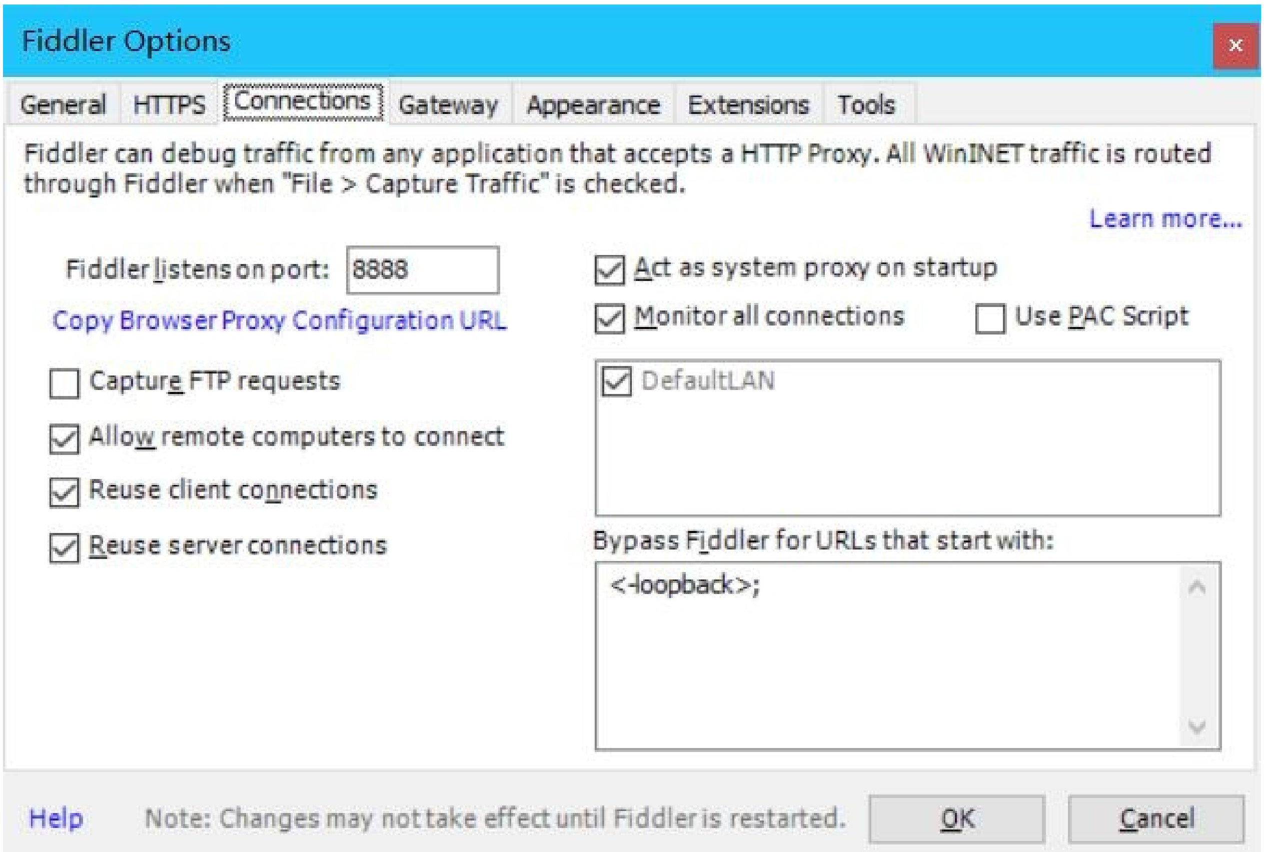
有些时候，我们只是想要获得 Web API。这种情况下，就不需要反编译 APK，直接抓取产生的封包即可。

这里我使用 Fiddler 演示一下如何抓取封包。

首先在这里下载 `Fiddler2`：https://www.telerik.com/download/fiddler/fiddler2

这个程序需要 .net 2.0 框架，Win7 之后自带，XP 的用户请到这里下载安装。

安装完成之后打开，访问菜单栏的 `Tools->Fiddler Options`，在弹出的窗口中选择 `Connections` 选项卡：



左上方的那个框就是端口，设置成与本机其它进程不冲突的端口号，我这里是 `8888`。

在电脑上打开控制台，执行 `ipconfig` 查看 IP：

```
C:\Users\asus> ipconfig

Windows IP 配置

...

无线局域网适配器 WLAN:

   连接特定的 DNS 后缀 . . . . . . . :
   本地链接 IPv6 地址. . . . . . . . : fe80::90ab:67f0:5c:78d9%7
   IPv4 地址 . . . . . . . . . . . . : 192.168.1.6
   子网掩码  . . . . . . . . . . . . : 255.255.255.0
   默认网关. . . . . . . . . . . . . : 192.168.1.1
```

我这里是`192.168.1.6`。

然后打开手机或者模拟器。（如果是手机的话，需要连接到同一个路由器。模拟器的话，位于同一个电脑上。）访问`设置->WLAN`，并长按当前连接的 WIFI：


2:21
WLAN
打开
WiredSSID
已连接
WiredSSID
取消保存网络
修改网络

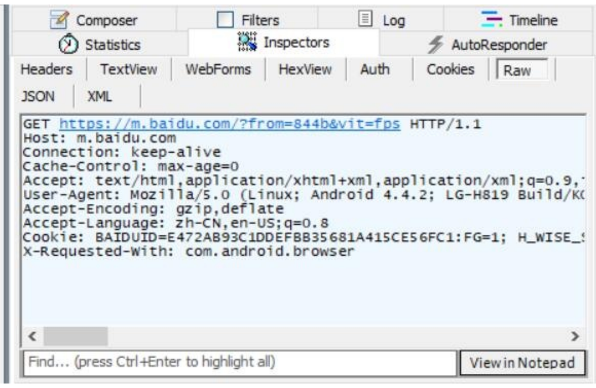
弹出对话框后，点击 `修改网络` ：



勾选 `显示高级选项` ，将“代理”设为 “手动”，并在主机名中填写上面的 IP，端口中填写上面设置的端口，并点击保存：



之后打开安卓这边的浏览器，随便访问一个网站，然后观察 Fiddler 的窗口：



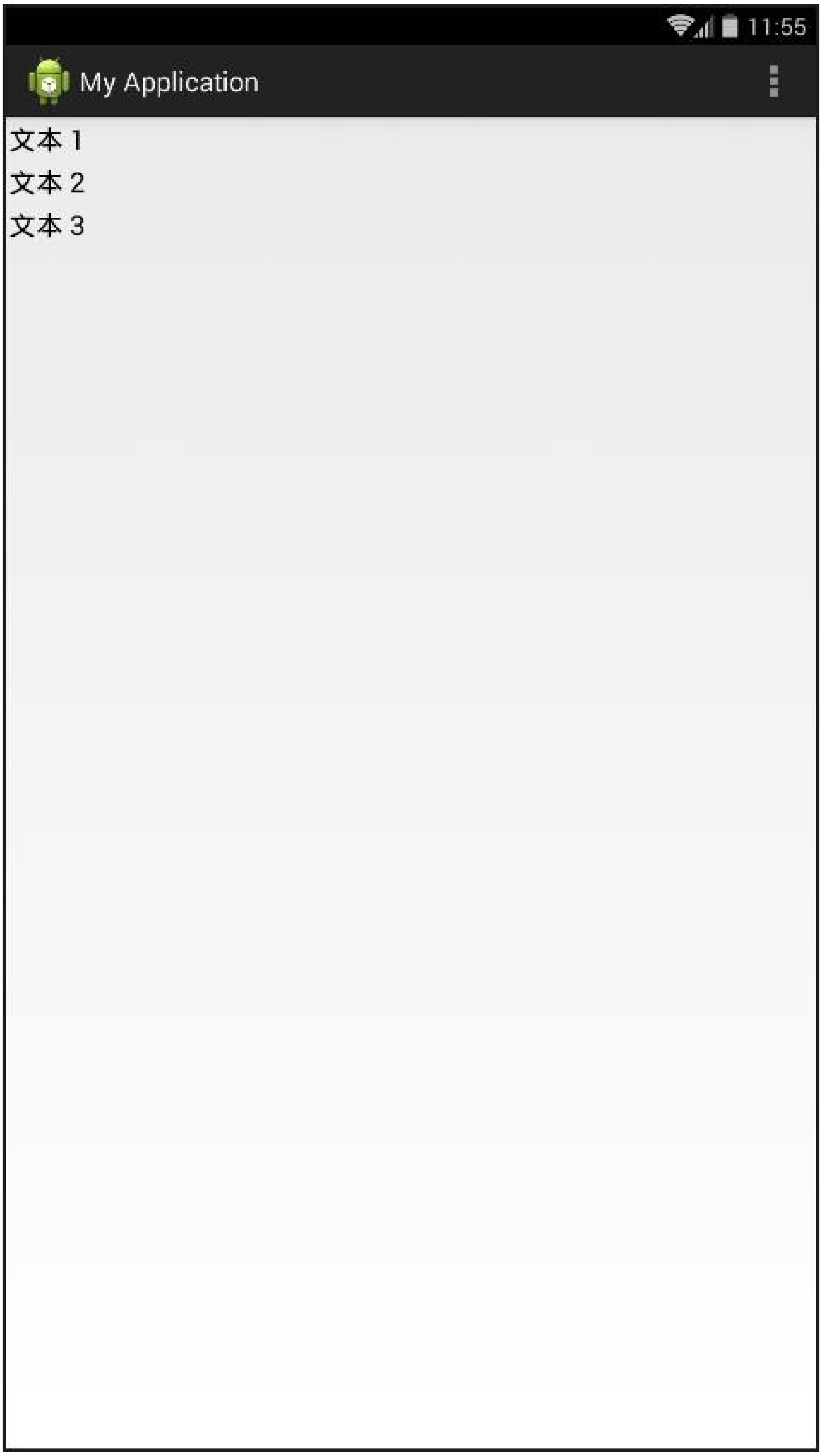
#	Result	Protocol	Host	URL	Body	Cach
558	200	HTTP	Tunnel to	m.baidu.com:443	779	
559	200	HTTPS	m.baidu.com	/?from=844b&vit=fps	26,213	no-ca
560	200	HTTP	Tunnel to	hm.baidu.com:443	781	
561	200	HTTPS	m.baidu.com	/static/index/plus/plus_log...	4,493	max-
562	200	HTTP	Tunnel to	gss0.bdstatic.com:443	0	
563	200	HTTP	Tunnel to	ss0.bdstatic.com:443	0	
564	200	HTTP	Tunnel to	gss0.bdstatic.com:443	0	
565	200	HTTPS	m.baidu.com	/static/index/plus/public/t...	2,259	max-
566	200	HTTP	Tunnel to	ss0.baidu.com:443	0	
567	200	HTTP	Tunnel to	ss1.baidu.com:443	0	
568	200	HTTP	Tunnel to	gss0.bdstatic.com:443	0	
569	200	HTTP	Tunnel to	ss1.baidu.com:443	0	
570	200	HTTP	Tunnel to	ss2.baidu.com:443	779	
571	200	HTTP	Tunnel to	ss0.baidu.com:443	0	
572	200	HTTP	Tunnel to	ss0.baidu.com:443	0	
573	200	HTTP	Tunnel to	ss1.baidu.com:443	779	

安卓逆向系列教程（四）实战篇

4.1 字符串资源

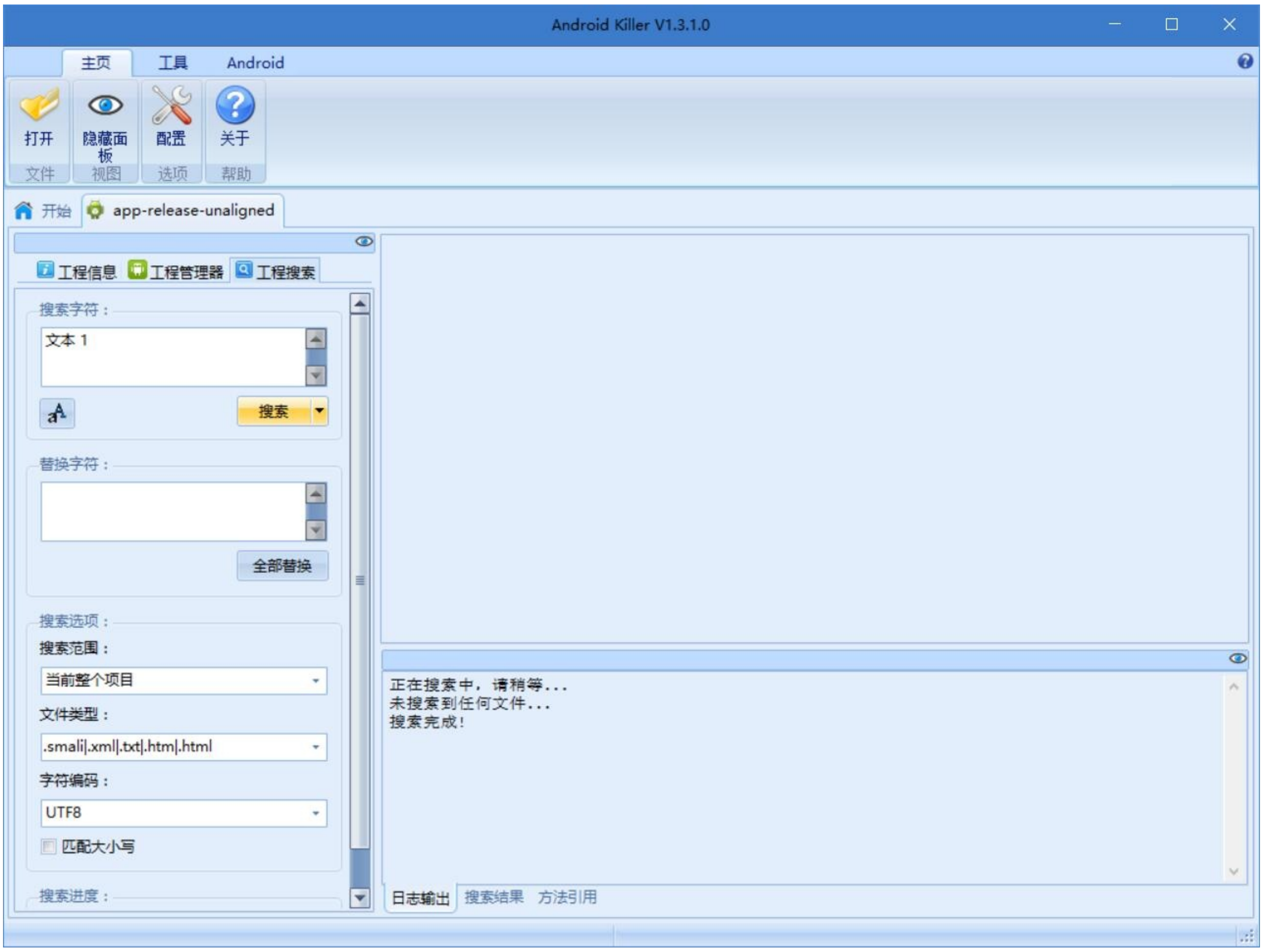
> 作者：飞龙

这篇教程是 APK 逆向实战的第一个例子，我会以一个非常简单的程序开始。主要内容就是修改字符串资源，除了破解所需之外，汉化也需要了解这个东西。我们的程序是这个样子。

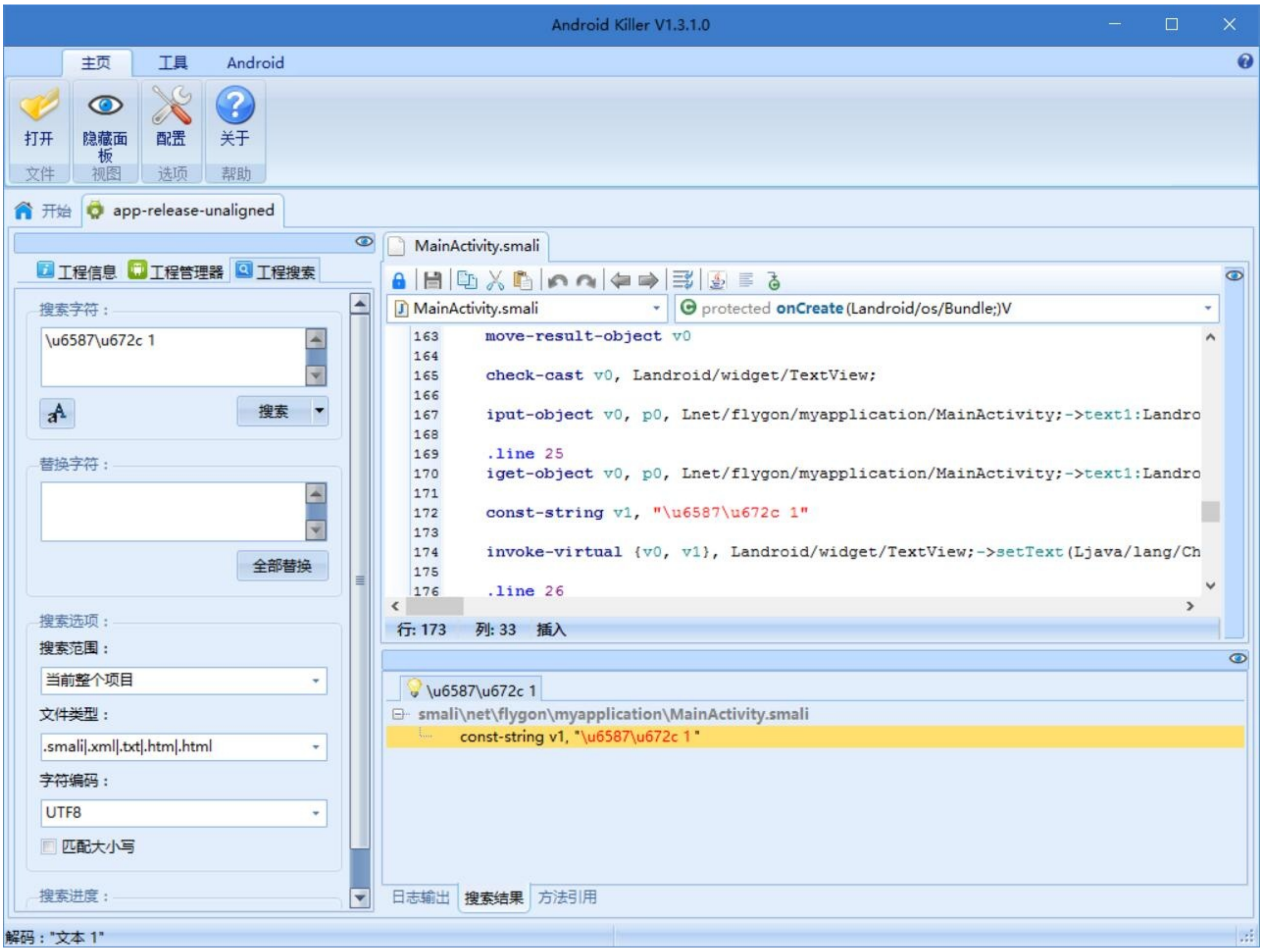


可以告诉大家的是，这三个文本的位置都不一样。

下面我们将其载入 Android Killer。完成后，在文本搜索框中搜索 `文本 1` 。

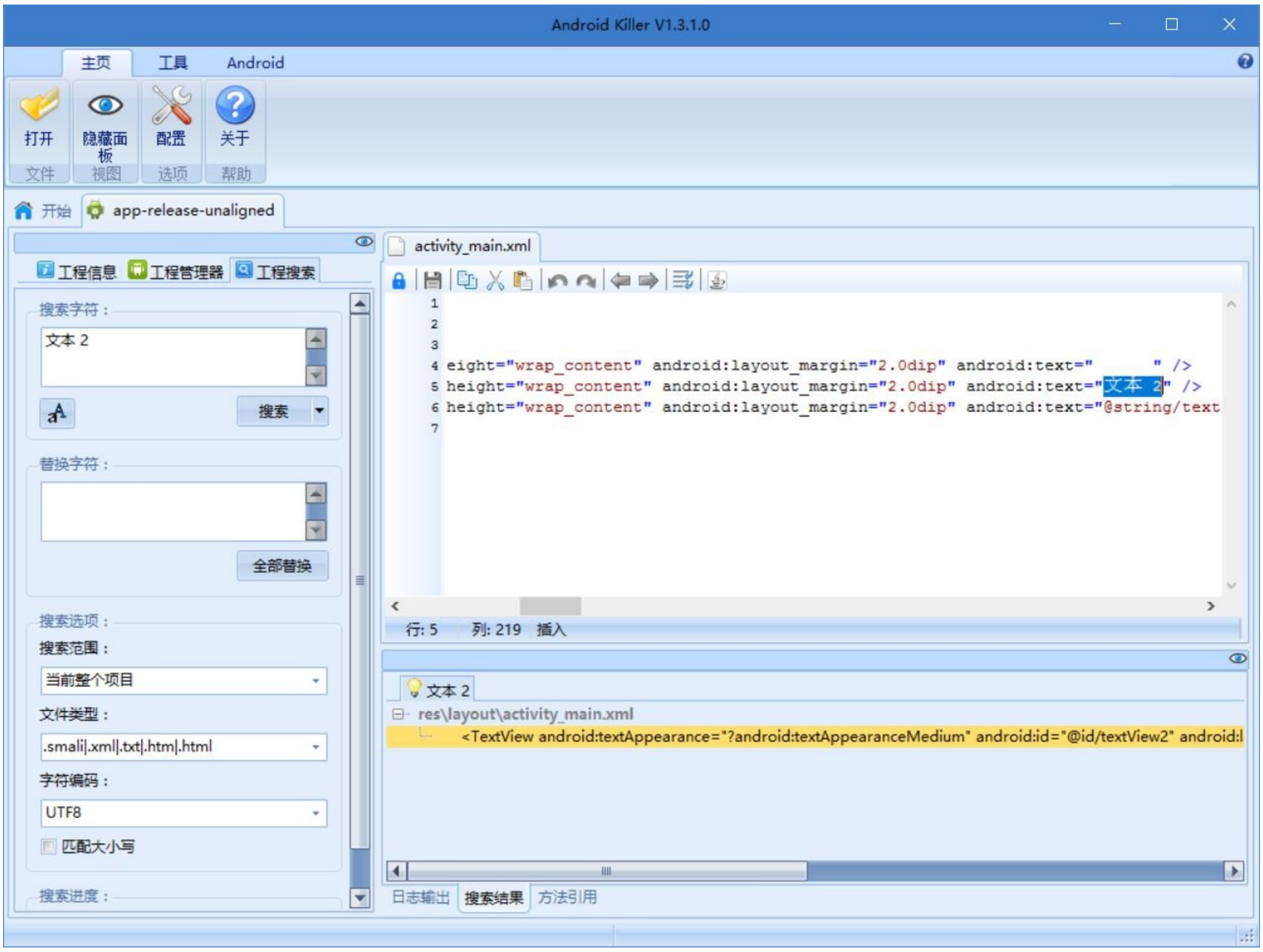


这样是不会有任何效果的，我们就猜测这个字符串应该是写入代码中的，而反编译出来的代码中的字符串以 `\uxxxx` 编码。所以我们要搜索 `\u6587\u672c 1` 。可以看到它的确存在于代码中。



下面我们要寻找 `文本 2` ，我们首先看一看这个函数，这是 `MainActivity` 的 `onCreate` 。这里没有其它的字符串了，说明一定在别处。

我们搜索 `\u6587\u672c 2` ，也是无效果的。那么我们搜索 `文本 2` 。

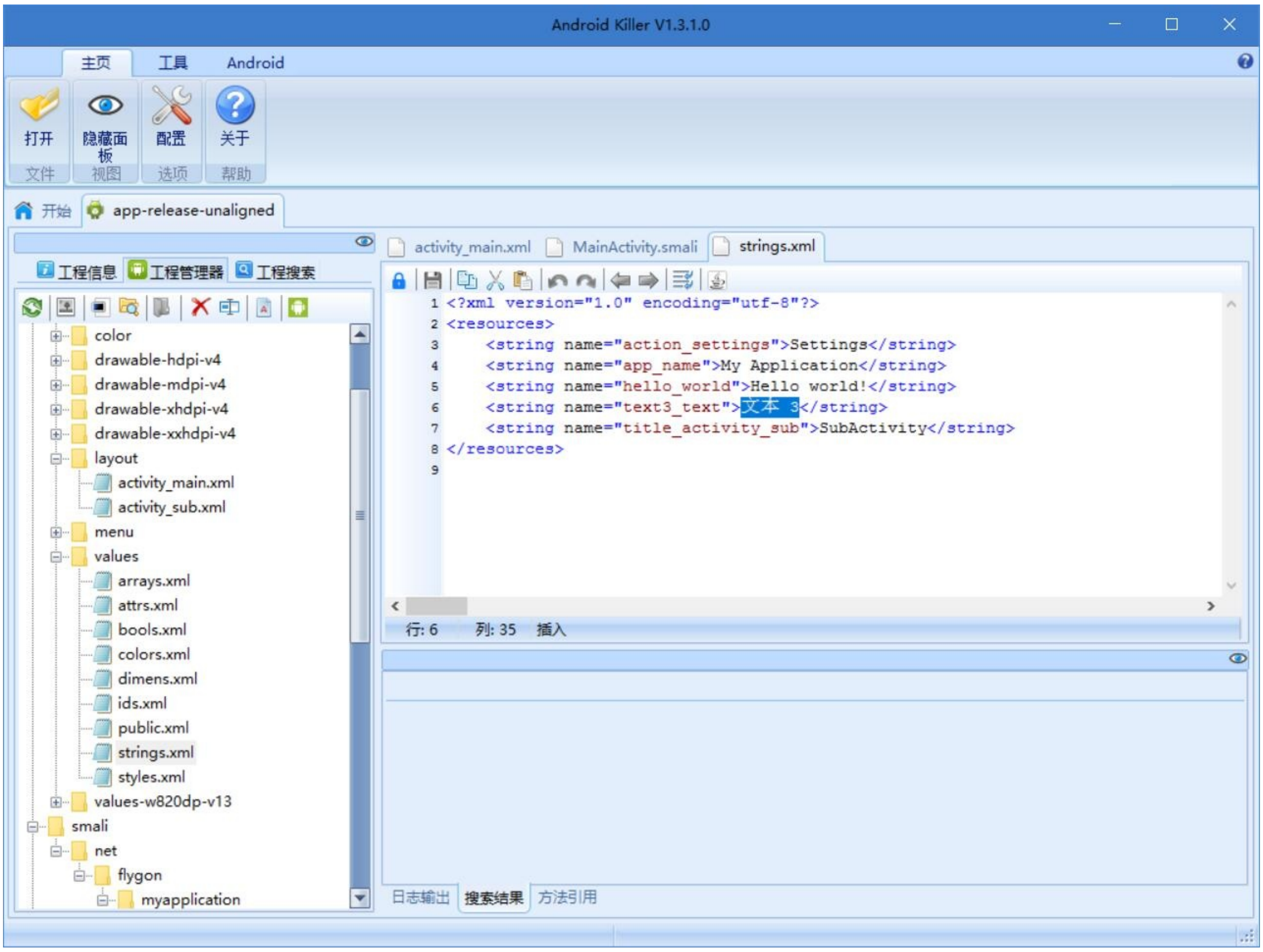



```
<TextView android:textAppearance="?android:textAppearanceMedium"
android:id="@id/textView" android:layout_width="wrap_content" an
droid:layout_height="wrap_content" android:layout_margin="2.0dip"
 android:text="       " />
<TextView android:textAppearance="?android:textAppearanceMedium"
android:id="@id/textView2" android:layout_width="wrap_content" a
ndroid:layout_height="wrap_content" android:layout_margin="2.0di
p" android:text="文本 2" />
<TextView android:textAppearance="?android:textAppearanceMedium"
android:id="@id/textView3" android:layout_width="wrap_content" a
ndroid:layout_height="wrap_content" android:layout_margin="2.0di
p" android:text="@string/text3_text" />
```

我们在 `activity_main` 里面找到了这个东西，它是 `MainActivity` 的布局文件，布局文件中的字符串是不编码的。所以以后我们就需要两种情况都试一试。

我们查看第三个 `TextView` ，它的 ID 是 `@id/textView3` ，那么肯定就是我们要找的第三个文本框。我们可以看到它的 `text` 属性是 `@string/text3_text` ，说明它可能在 `strings.xml` 里面。

我们直接访问 `strings.xml` ，我们可以看到 `文本 3` 在这里：



虽然 Android 不提倡硬编码在代码或者布局文件里面，但总有些人是这样做的，我们就需要了解。

4.2 ������������

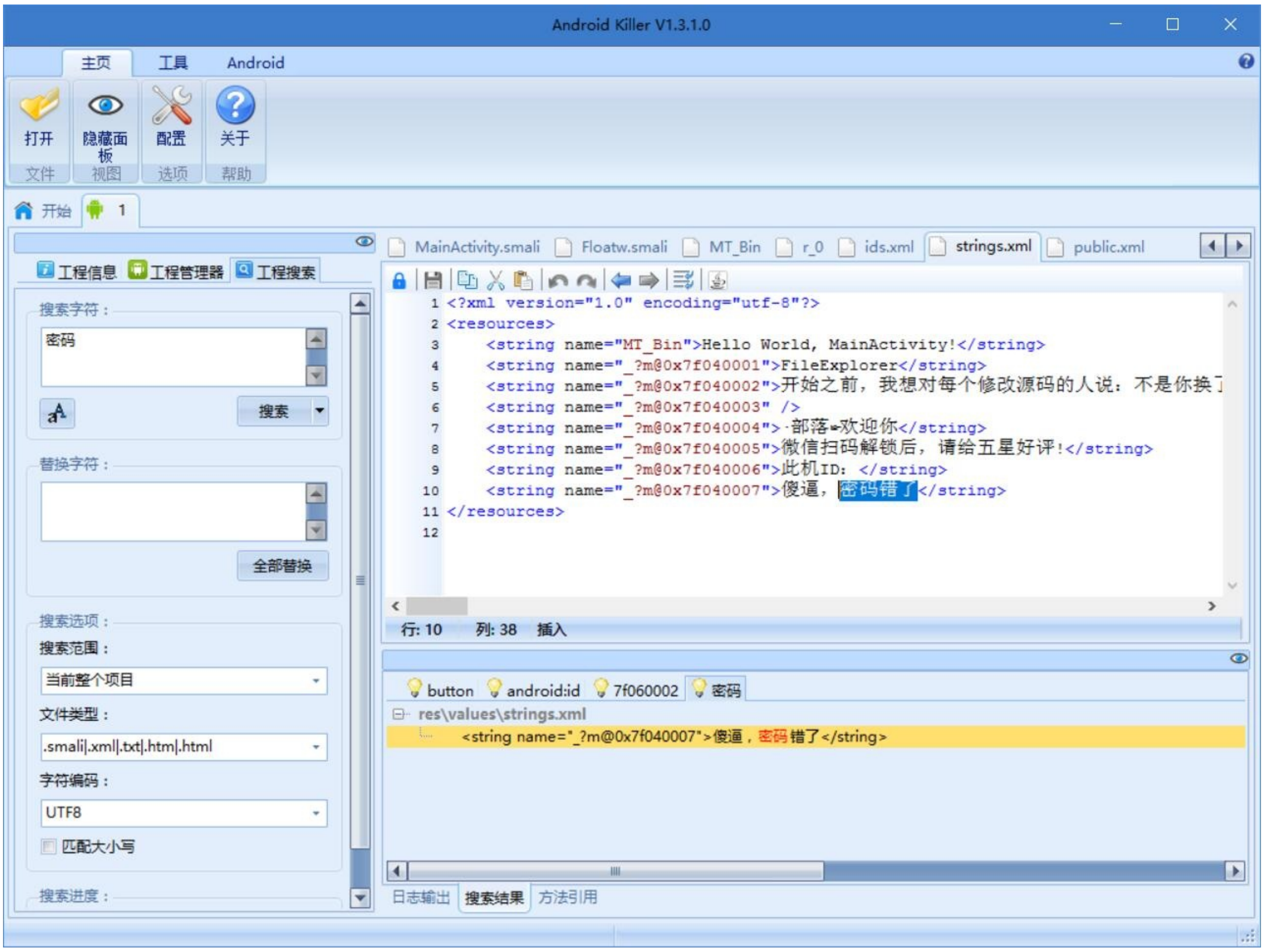
> ���□�����

�����̲������Ç����ₕ���������������������
�������Łۄ ���ö�����ֻ������ڿ���ʒ���������
�ݔ���ùڀ��������λ��

�������������Ľ�ɕ��������������������
�ǒ �□����ֻ�ģ���������û�취�ص����档


夜神安卓模拟器
时空先生
·部落☞欢迎你
微信扫码解锁后，请给五星好评!
此机ID：108316
扫码支付
微信扫一扫转账
微信支付自动解锁屏幕
自动解锁
提示：微信扫描二维码

[illegible]



[illegible] `_?m@0x7f040007` [illegible] pu [illegible] `0x7f040007` [illegible] 2130 [illegible]

```
<public type="string" name="_?m@0x7f040007" id="0x7f040007" />
```

[illegible] Java [illegible]

```
paramAnonymous2View = (TextView)Floatw.access$L1000002(Floatw.this).findViewById(2131099651);
paramAnonymous2View.setText(Floatw.this.getResources().getString(2130968583));
```

[illegible]

[illegible]

```
@Override
public void onClick(View paramAnonymous2View)
{
    int i = Floatw.this.my_password;
    if (this.val$etext.getText().toString().equals(String.valueO
f(i)))
    {
        paramAnonymous2View = Floatw.this;
        Floatw localFloatw = Floatw.this;
        try
        {
          Class localClass = Class.forName("com.as.xiaoyu.Floatw"
);
          paramAnonymous2View.stopService(new Intent(localFloatw
, localClass));
          return;
        }
        catch (ClassNotFoundException paramAnonymous2View)
        {
          throw new NoClassDefFoundError(paramAnonymous2View.get
Message());
        }
    }
    // �ηŵĴ���
    // ...
}
```

���� `val$etext` ������ǿ���� `i` ���_���_�����ô�ɔ���ɹ 2�������ͼ����������¿�������� ̤ ���������� ��������ᵒ ����ʹ������ һ������ �����лȡ�� `WindowsM` ��Ȼ���������ɻ�� `LinearLayout` ��������������ʆ�� �������Ч�����

```
private void createFloatView()
{
    this.wmParams = new WindowManager.LayoutParams();
    Application localApplication = getApplication();
    this.mWindowManager = ((WindowManager)localApplication.getSy
stemService(Context.WINDOW_SERVICE));
    this.wmParams.type = 2010;
    this.wmParams.format = 1;
    this.wmParams.flags = 1280;
    this.wmParams.width = -1;
    this.wmParams.height = -1;
    this.mFloatLayout = ((LinearLayout)LayoutInflater.from(getAp
plication()).inflate(2130903041, (ViewGroup)null));
    this.mWindowManager.addView(this.mFloatLayout, this.wmParams
);
    this.mFloatLayout.measure(View.MeasureSpec.makeMeasureSpec(0
, 0), View.MeasureSpec.makeMeasureSpec(0, 0));
}
```

Ӽ���� ˘ ������'����� ɹq��Ǿɔ������������

���Dz `onClick` �ĵ�һ�У����� `i` �� `my_password` ��Ȼ��������� ��

```
int my_password = this.number * 2 + 1;
int number = (int)((Math.random() + 1) * 100000);
```

������������酞ⵑ
���������̤�� `number` � ˝ ��ټ�һ������ `number` ��ʲô�ĺ ����������� `number` ��

```
Object localObject = (TextView)Floatw.access$L1000002(Floatw.this
).findViewById(2131099650);
String str = Floatw.this.getResources().getString(2130968582);
((TextView)localObject).setText(str + String.valueOf(Floatw.this
.number));
```

��Щ������������һ���ı������̤������Ȼ������̤� ���� `number` у�������'������������̤�������� ������̀Ψ�н��� `�̤�ID��` �����ô `number` ��Ö����Å���� ���� ̥ ���������� `108316` ����ô��������Ö����� `216` ��

������һ�����昰

```
localObject = (EditText)Floatw.access$L1000002(Floatw.this).find
ViewById(2131099649);
```

`2131099649` ��Ų�������Ï `0x7f060001` ��������ȩ� ♀ ���Ǹ ����������

```
<EditText android:textColor="#ff000000" android:id="@id/_?m@0x7f
060001" android:background="#ffffffff" android:layout_width="200
.0dip" android:layout_height="4.0dip" android:layout_marginEnd="
200.0dip" />
```

��� ����������һЩ���⁞û����� `val$etext` ��͇�Ы�͈�� �������Ö�þ��������� �□������������������ ц��������Ҳ����Ѣ������ǿ���Ч��������� ����к□Ặ�С ������ǻ���п���ȣ��������ڑ�ť����·���Ĵ����λ����є�һ��George ����Ͳ�����˥�


夜神安卓模拟器
时空先生
·部落☞欢迎你
微信扫码解锁后，请给五星好评!
此机ID：197858
扫码支付
微信扫一扫转账
微信支付自动解锁屏幕
自动解锁
提示：微信扫描二维码
17:20

���ǰψ˷���������ö���ɼ����Ȼ������� `216633` ���
ʿε����������ť��


夜神安卓模拟器
5:20
Hello World, MainActivity!
17:20

�����ɹ����������ǿ�� ��·��پ����˳��ˈ�

�ο�

- ��������Ղ��21��������� - ������������

4.3 登山赛车内购破解

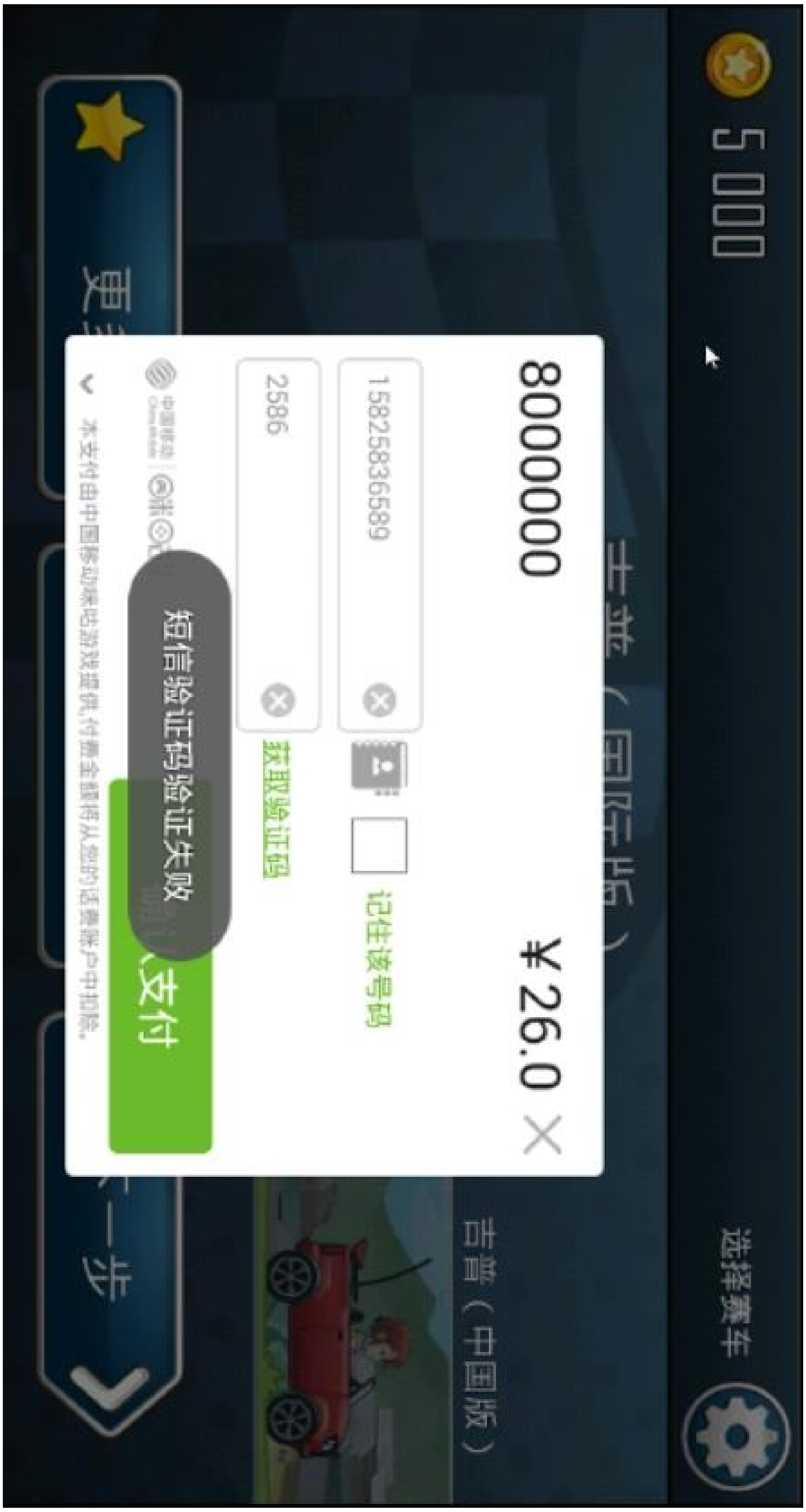
> 作者：飞龙

首先在这里下载游戏：http://g.10086.cn/game/760000032287?spm=www.pdindex.android.addjgame.1

我们要破解的东西是这个，获得金币：

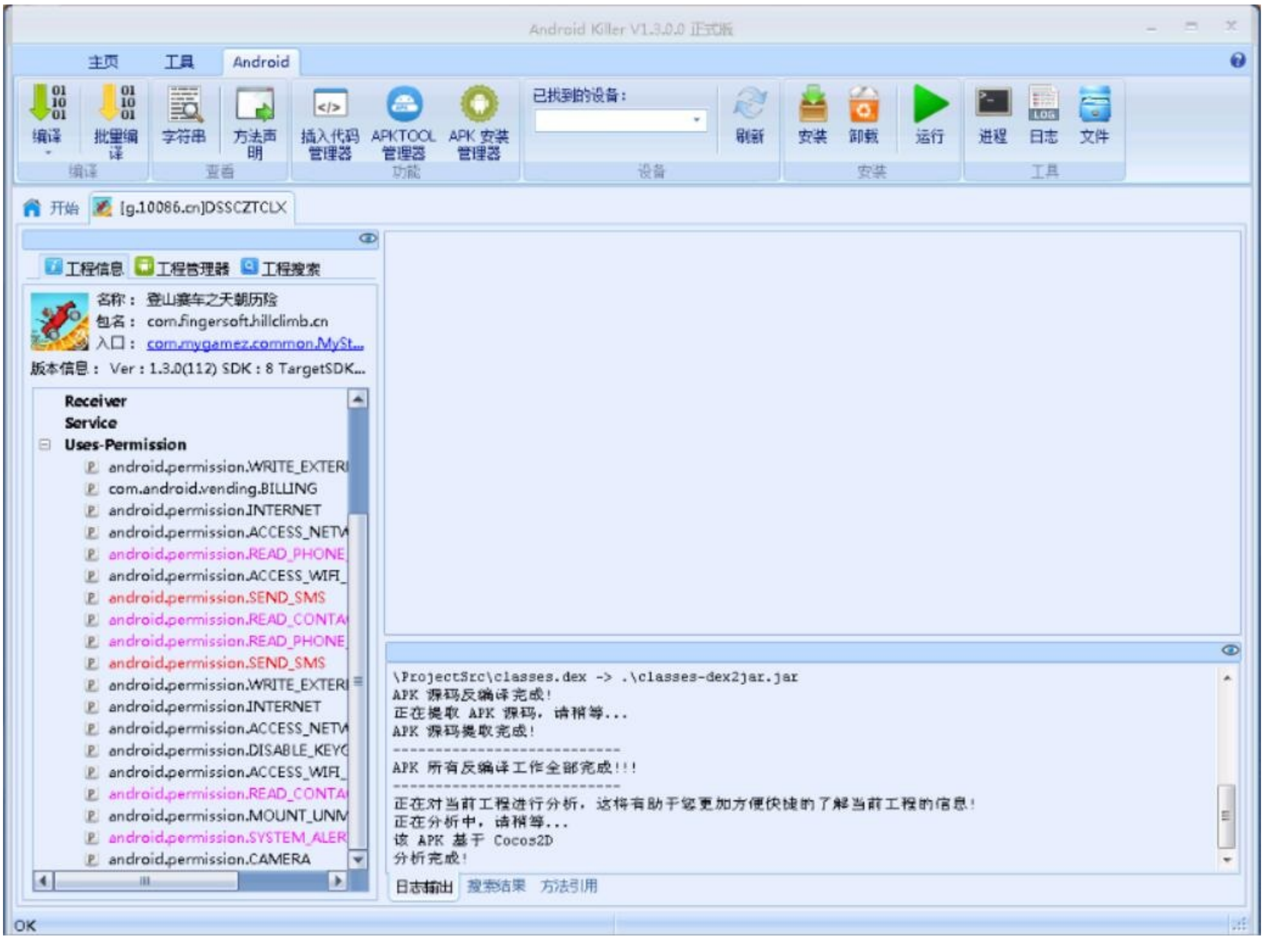


点击之后会有个弹出框，我们随便输入一些东西，然后点击“确认支付”：

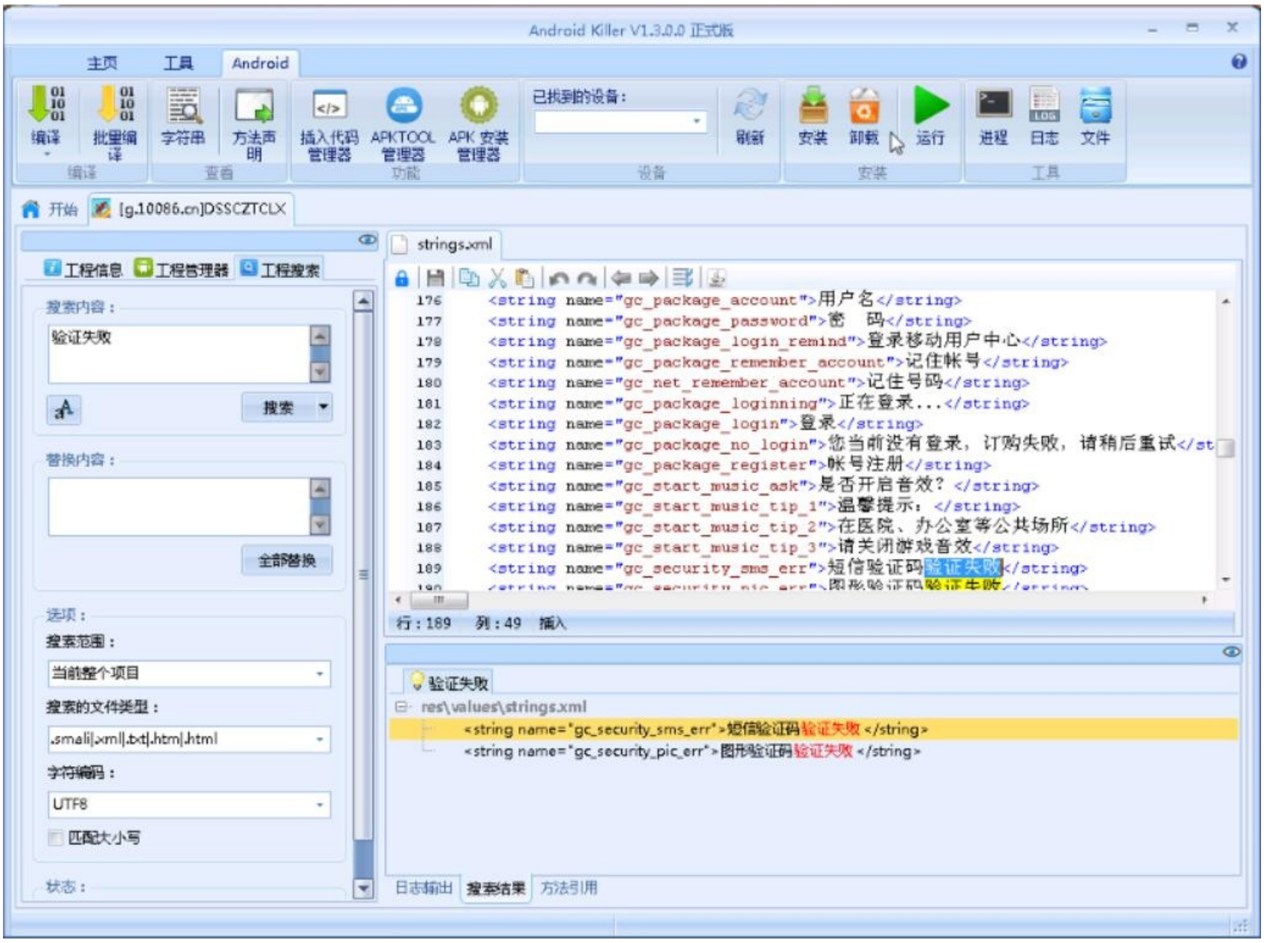


出现了“短信验证码验证失败”的 `Toast` 。

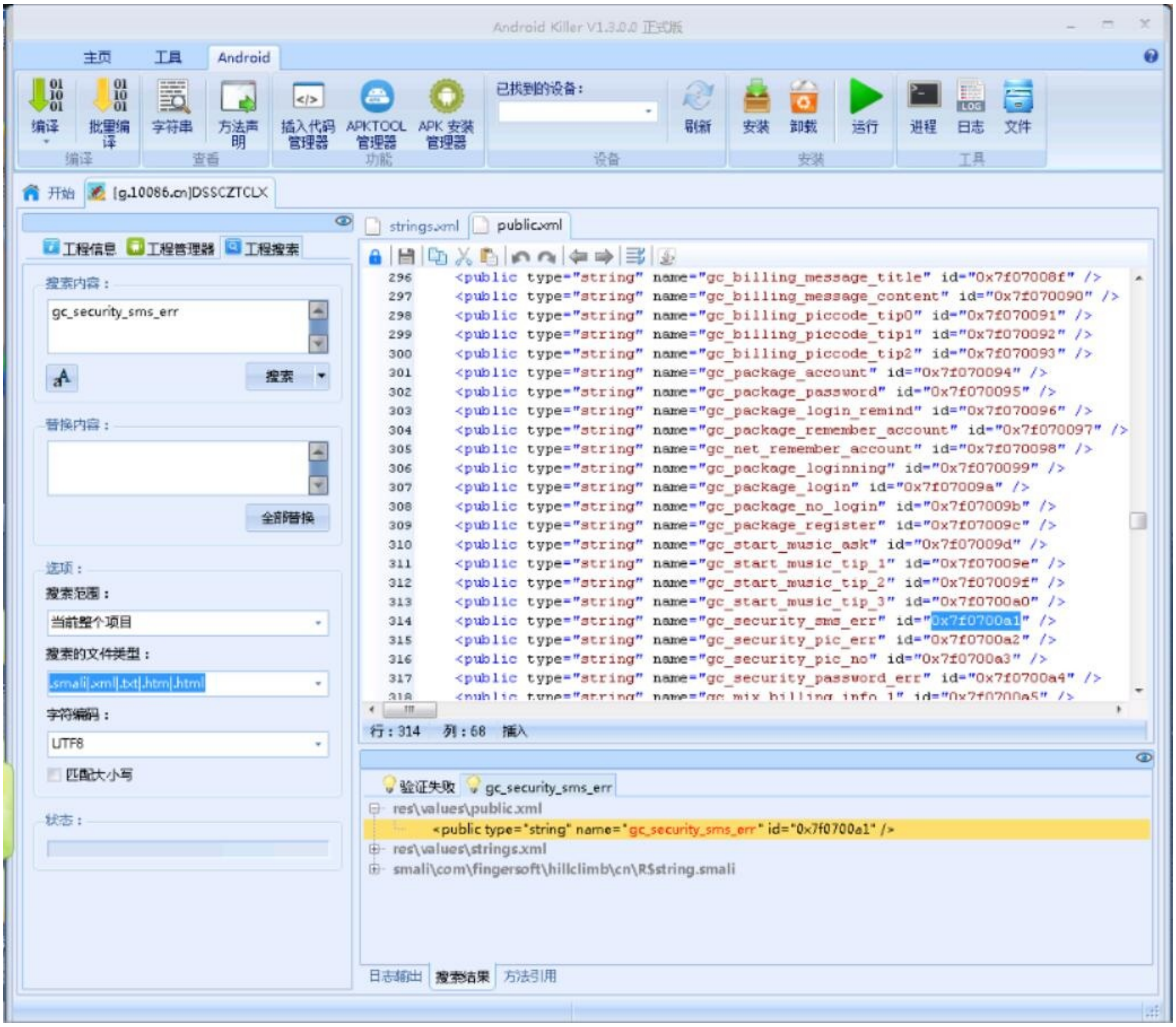
好，信息收集完毕，将程序拉进 Android Killer：



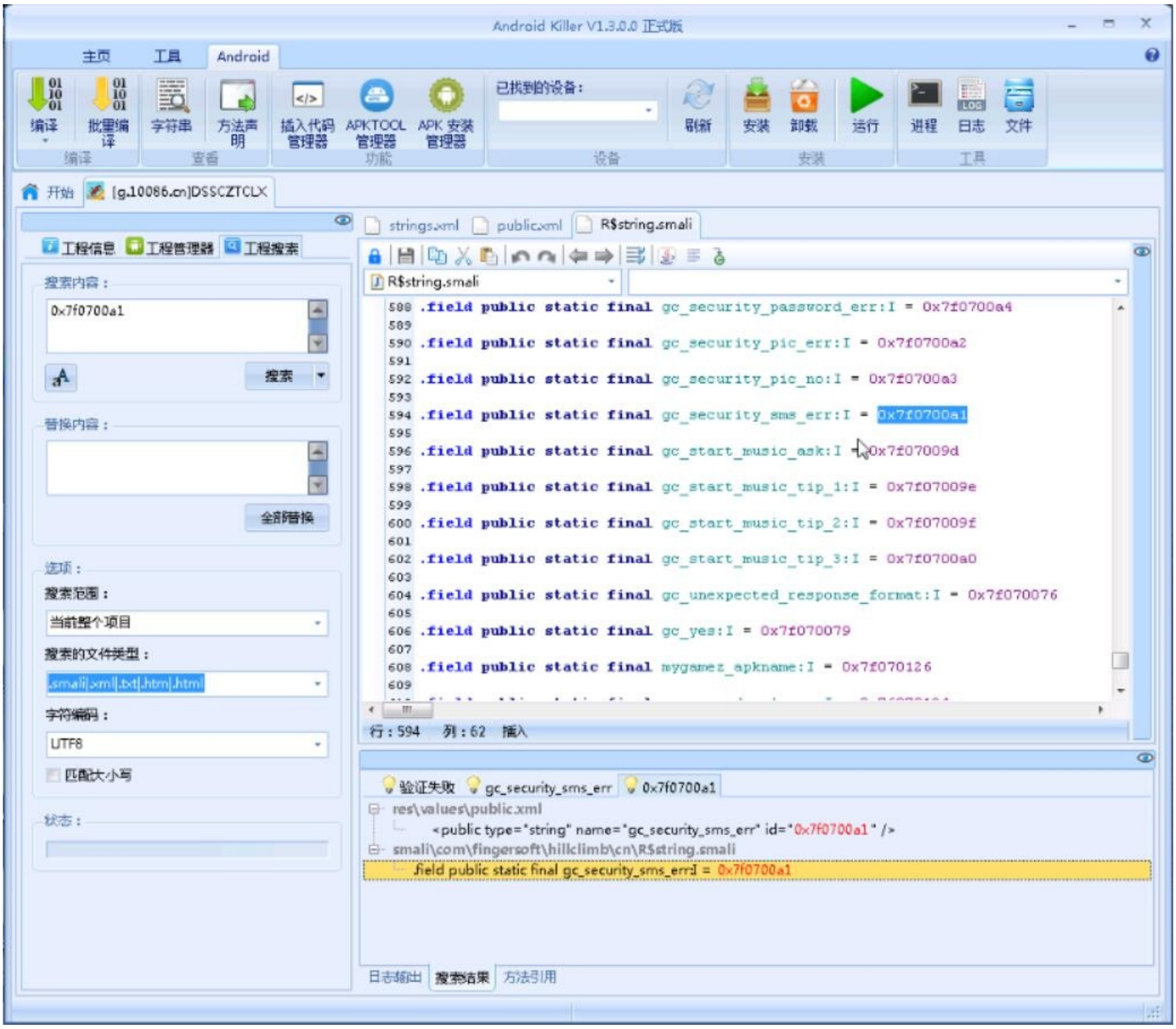
搜索“验证失败”四个字，我们可以找到刚才的内容：



我们发现它在 `strings.xml` 里面，它的名称是 `gc_security_sms_err` 。老方法，搜索这个名称：

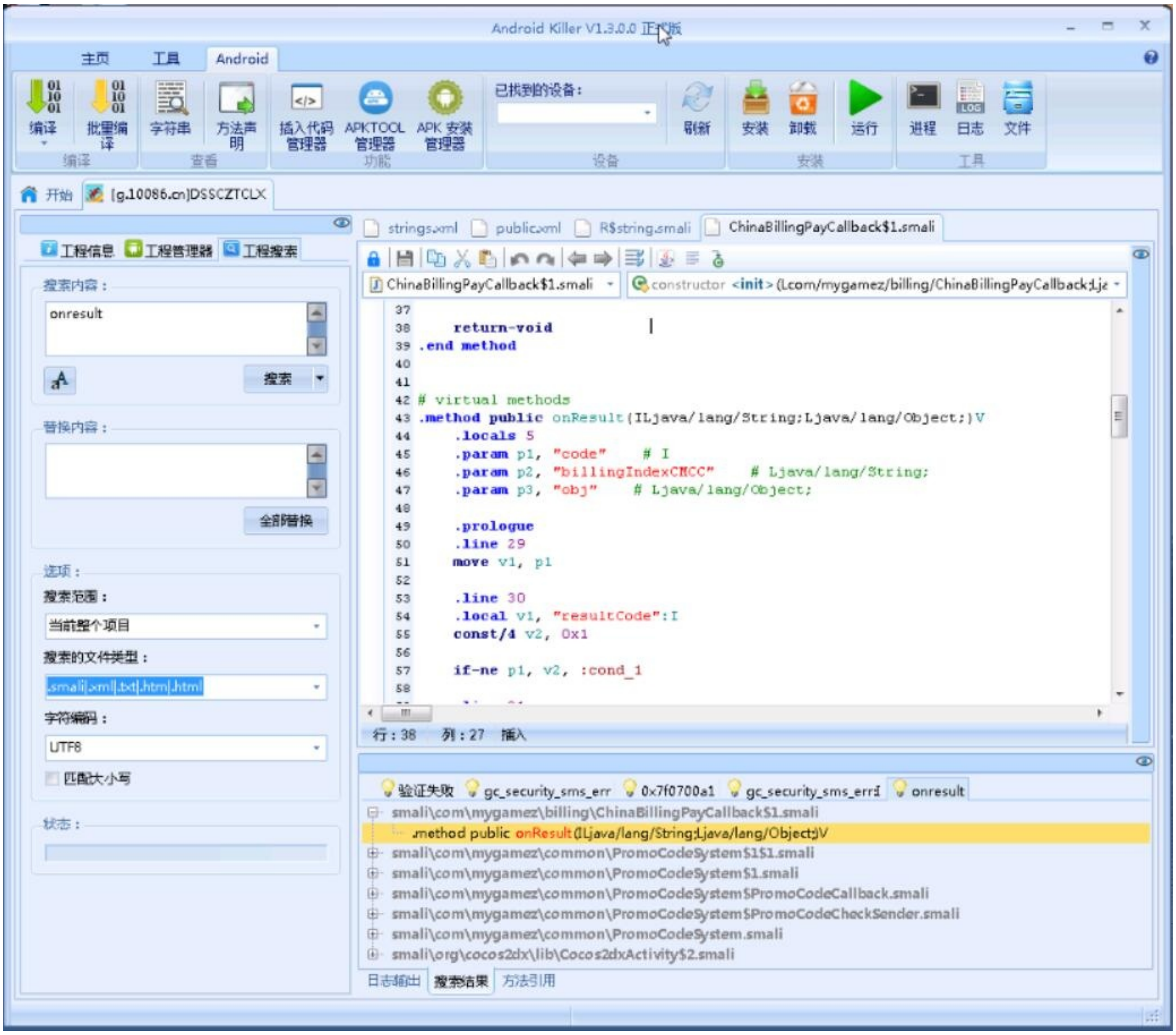


我们在 `public.xml` 中发现了它的 ID，`0x7f0700a1`。我们搜索这个值。



然后就没下文了。我们没有找到任何使用这个值的地方。只能从其它方面入手。

我们从前面可以得知，付费用的是移动的接口，我们搜索 `onresult` ，这是移动支付 API 的关键字（问我怎么知道的，这个 API 是有开发者文档的，大家可以搜索一下）：



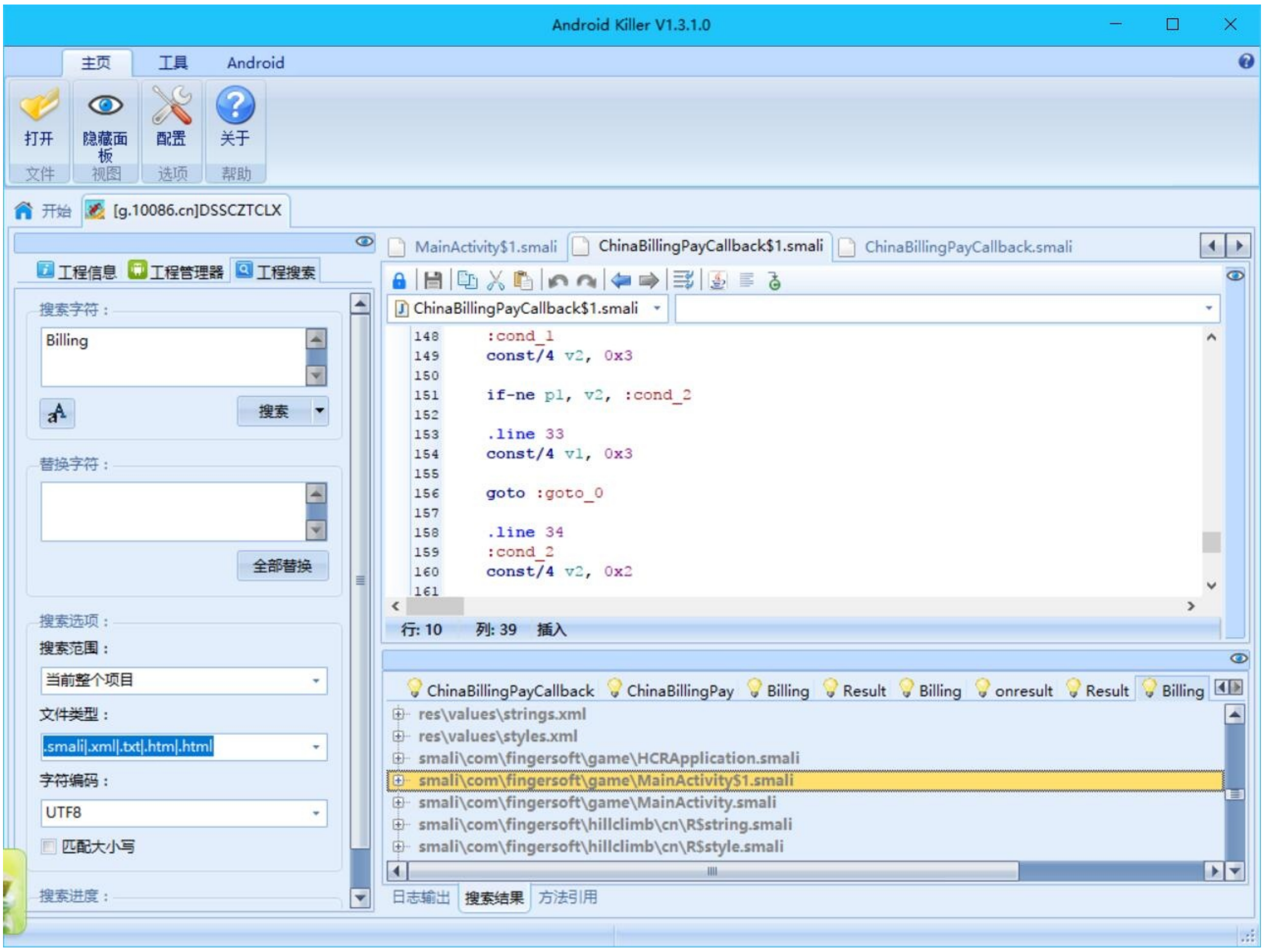
一下子就出现了，`ChinaBillingPayCallback$1`。下面就是要好好分析这个类

这个类只有一个`onResult`方法，也就是说只是一个闭包，而且也没有什么特别有用的信息：

```
public void onResult(int paramInt, String paramString, Object paramObject)
{
    int i = paramInt;
    BillingResult localBillingResult = new BillingResult();
    localBillingResult.setResultCode(i);
    localBillingResult.setBillingIndex(this.val$billingIndex);
    localBillingResult.setReturningObject(paramObject);
    localBillingResult.setCode(paramString);
    Log.i("MySDK Billing Java", "CMCC object toString(): " + paramObject.toString());
    Log.i("MySDK Billing Java", "CMCC result: " + localBillingResult.toJSON());
    this.this$0.launchResultReceived(localBillingResult);
}
```

反正我是没看出来。再看一看，这个类所在的包是 `com.mygamez.common` ，而软件的包是 `con.fingersoft.game` ，等于说这个类是别人的 API，当然没有业务逻辑。

我们换个方式，搜索 `Billing` ：



除了底下的两个资源类，有三个游戏包中的类出现了 `Billing` 。第一个类是一个 `Applicaion` ，只有如下代码：

```
public void onCreate()
{
    MyBilling.applicationHeater(this);
}
```

再看看 `MainActivity$1` ，它也是一个闭包，只有 `onChinaBillingResult` 方法，这就非常重要了。由于它的 java 反编译结果不可读，我们直接看 Smali：

```
invoke-virtual {p1}, Lcom/mygamez/billing/BillingResult;->getRes
ultCode()I

move-result v2

packed-switch v2, :pswitch_data_0

...
```

开头有这么一段代码，我们跳到 `:pswitch_data_0` 处：

```
:pswitch_data_0
.packed-switch 0x1
    :pswitch_0
    :pswitch_1
.end packed-switch
```

由于判断的是某个 API 的返回代码，按照惯例， `0` 是正常，其余是异常。我们可以跳到 `:pswitch_1` 分支：

```
:pswitch_1
iget-object v2, p0, Lcom/fingersoft/game/MainActivity$1;->this$0:
Lcom/fingersoft/game/MainActivity;

invoke-virtual {v2}, Lcom/fingersoft/game/MainActivity;->getAppl
icationContext()Landroid/content/Context;

move-result-object v2

const-string v3, "\u4ed8\u6b3e\u5931\u8d25"

const/4 v4, 0x1

invoke-static {v2, v3, v4}, Landroid/widget/Toast;->makeText(Lan
droid/content/Context;Ljava/lang/CharSequence;I)Landroid/widget/
Toast;

move-result-object v1

.line 301
.local v1, "toast":Landroid/widget/Toast;
invoke-virtual {v1}, Landroid/widget/Toast;->show()V

goto :goto_0
```

那个字符串是“付款失败”，这么多代码其实就是`Toast.makeText(this.this$0.getApplicationContext(), "付款失败", 1).`。这应该就是异常分支了。

至于接下来的修改，这个方法第二行`move-result v2`，改成`const v2, 0x0`，完事。

> 写到这里其实还有一个问题，就是代码中的字符串和实际实现的对不上。这个我也不知道为什么，但是既然有这种情况，就要想别的方法。

4.4 逆向云播 VIP

> 作者：飞龙

这次的软件是刀哥云播，在这里下载：http://www.xuepojie.com/thread-23860-1-1.html

我们先分析一下行为：

打开之后的界面是这样，并弹出一行消息：



我们先点击"福利"，弹出这样一行字。



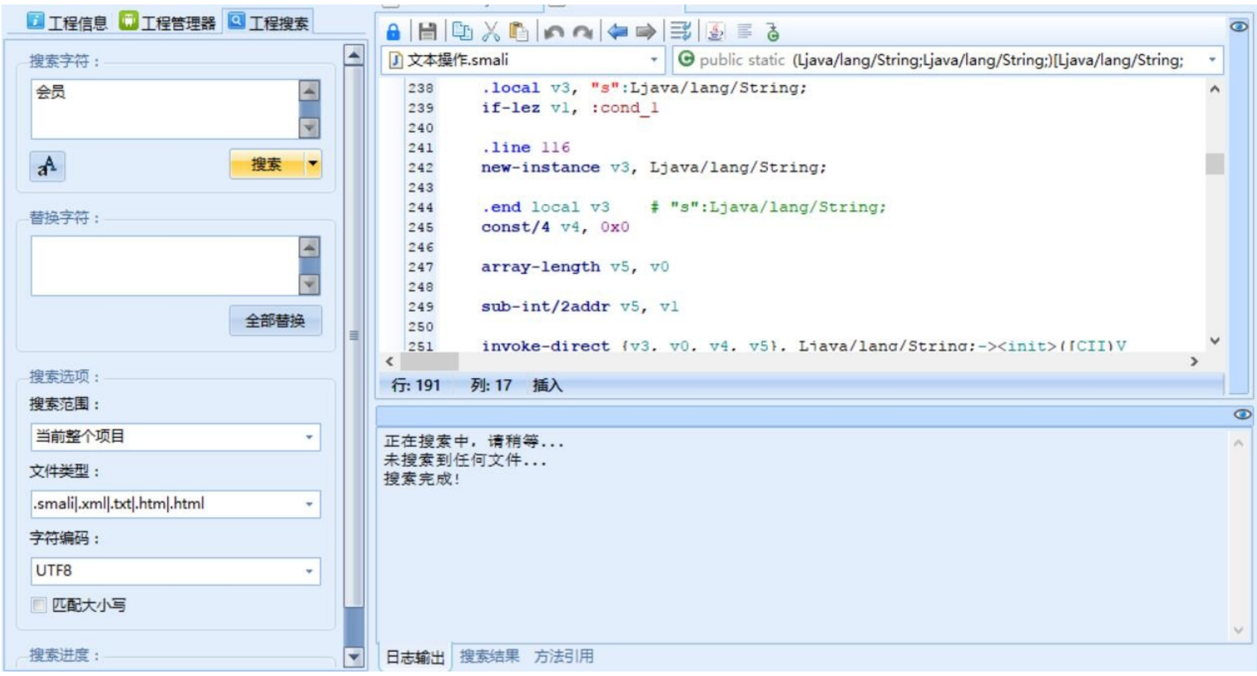
然后我们点击那个"设置"，在设置界面上，我们如果点击"加入会员"，弹出这样一行消息。



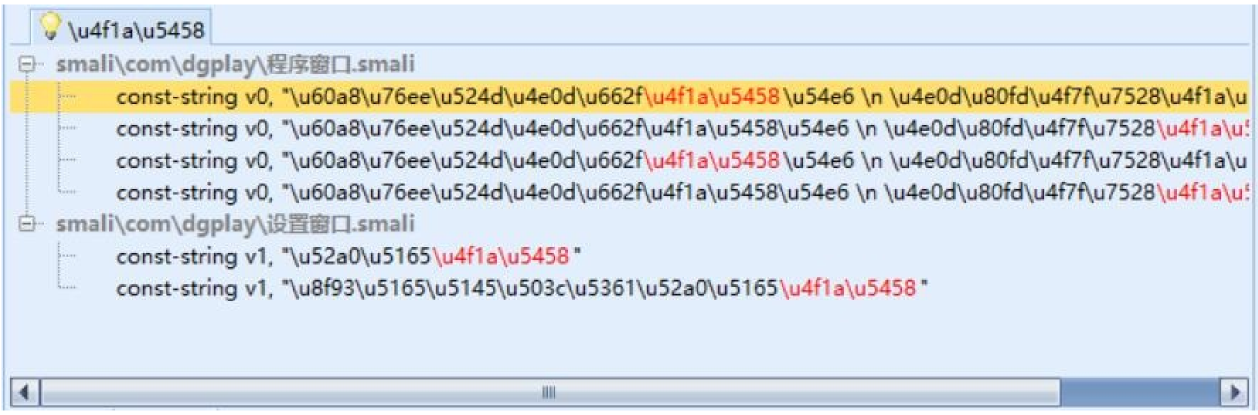
然后我们将其拖进 Android Killer，发现它是 e4a 编写的。



我们直接搜索“会员”，没有任何结果。可见 e4a 直接将字符串硬编码进代码里。



我们搜索它的 Unicode 编码， `\u4f1a\u5458` 试试看：



一共有四个。`程序窗口`类中有两个，都在`图片列表框1$表项被单击`方法中。内容都是“您目前不是会员...”。设置窗口里的都是“加入会员”，是按钮的标题，和破解没太大关系。

我们点击第一个，文本附近的代码是这样：

```
.line 27
:cond_4
if-nez v0, :cond_1 # return-void

.line 28
const-string v0, "\u60a8\u76ee\u524d\u4e0d\u662f\u4f1a\u5458\u54e6 \n \u4e0d\u80fd\u4f7f\u7528\u4f1a\u5458\u529f\u80fd" # 您目前不是会员

invoke-static {v0}, Lcom/e4a/runtime/应用操作;->弹出提示(Ljava/lang/String;)V

goto :goto_0 # return-void
```

经分析可得，这个是失败分支的代码。然而`:cond_1`不是成功分支，它直接指向`return-void`，所以关键跳应该不在这里。由于安卓程序编译后的 Smali 是打乱的，应该在跳到`:cond_4`的地方。

我们在相同方法中寻找，只找到一处满足要求的地方：

```
# 项目索引
.param p1, "\u9879\u76ee\u7d22\u5f15"    # I

.prologue
const/4 v3, 0x1

.line 11
.line 12
if-nez p1, :cond_2
# p1 为 0 时的操作（随播）
# ...

.line 19
:cond_2
if-ne p1, v3, :cond_5
# p1 为 1 时的操作（短片）

.line 20
sget-boolean v0, Lcom/dgplay/公用模块;->vip:Z

.line 21
# 关键跳
if-ne v0, v3, :cond_4
# 启动短片窗口的代码
# ...
```

要注意这个回调对应图片列表框，就是主界面上的六个按钮（请见图 1）。其中 `p1` 是被选中项，按照惯例是从 0 开始，从左到右从上到下。我们刚才查看的这段代码是 `p1` 为 1 情况下的，也就是你点击右上角的按钮之后会触发。

（虽然我们实际上不推荐把六个回调都写到一个函数，因为它们是六个不同的逻辑，但是由于这是别人的代码，只能忍了。）

我们可以看到 `p1` 为 1 时，首先获取了 `com/dgplay/公用模块` 的 `vip` 静态字段，判断它是不是 `1` （ `v3` ），是的话就启动窗口，不是的话就弹出消息。

我们还有另一个提示不是会员的位置，也就是我们之前测试过的“福利”按钮（下标为 4）。我们点过去看看：

```
.line 41
:cond_9
const/4 v0, 0x4

if-ne p1, v0, :cond_c
# p1 为 4 时的代码

.line 42
sget-boolean v0, Lcom/dgplay/公用模块;->vip:Z

.line 43

# 关键判断
if-ne v0, v3, :cond_b
# 启动福利窗口
# ...

.line 48
:cond_b
if-nez v0, :cond_1 # return-void

.line 49
const-string v0, "\u60a8\u76ee\u524d\u4e0d\u662f\u4f1a\u5458\u54
e6 \n \u4e0d\u80fd\u4f7f\u7528\u4f1a\u5458\u529f\u80fd"

invoke-static {v0}, Lcom/e4a/runtime/应用操作;->弹出提示(Ljava/lang
/String;)V

goto/16 :goto_0 # return-void
```

果然还是根据这个字段。我们猜想这个静态字段是控制当前用户是否是 VIP 的字段。为了减少工作量，以及避免可能的暗桩，我们不要直接改动判断，而是将这个字段赋成 1。

但是，在静态构造器中将这个字段赋成 1 是不行的，因为主界面中可能有将它重新赋成 0 的代码。我们搜索 `Lcom/dgplay/公用模块;->vip` ，结果如下：



其中只有主窗口和设置窗口有赋值。设置窗口那个不用看了，因为是购买 VIP 的地方，它肯定是将其赋值为 1，那么主窗口中：

```
# 第一处
.line 16
const/4 v0, 0x1
sput-boolean v0, Lcom/dgplay/公用模块;->vip:Z

# 第二处
.line 18
:cond_0
const-string v0, "\u60a8\u76ee\u524d\u8fd8\u4e0d\u662fVIP,\u53ef
\u8d5e\u52a9\u4f5c\u8005\u83b7\u5f97VIP"

invoke-static {v0}, Lcom/e4a/runtime/应用操作;->弹出提示(Ljava/lang
/String;)V

.line 19
# v1 为 0x0
sput-boolean v1, Lcom/dgplay/公用模块;->vip:Z

goto :goto_0
```

这两处都在 `多线程1$取网页源码2完毕` 函数中。当然第二处是将其赋为 0 的地方，通过分析逻辑得知，这个是失败分支，而另一个是成功分支。我们向上找到关键跳转：

```
# 关键跳转
if-eqz v0, :cond_0

# ...
# 第一处
# ...
```

把这句去掉，就可以了。重新编译、打包，大功告成。

4.5 糖果星星达人

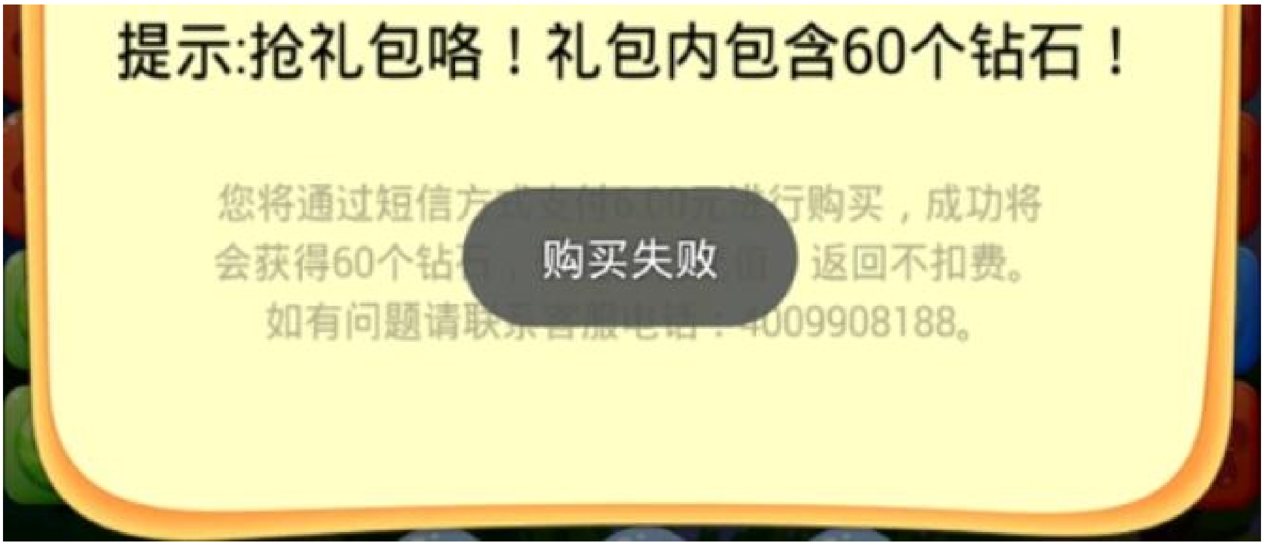
> 作者：飞龙

软件在这里下载：http://www.anzhi.com/soft_2539282.html

第一次进入游戏之后，会弹出来一个“新手礼包”，关掉之后，点击“新游戏”，之后进入“关卡1-1”。我们点击左上角的加号，会出现这个界面：



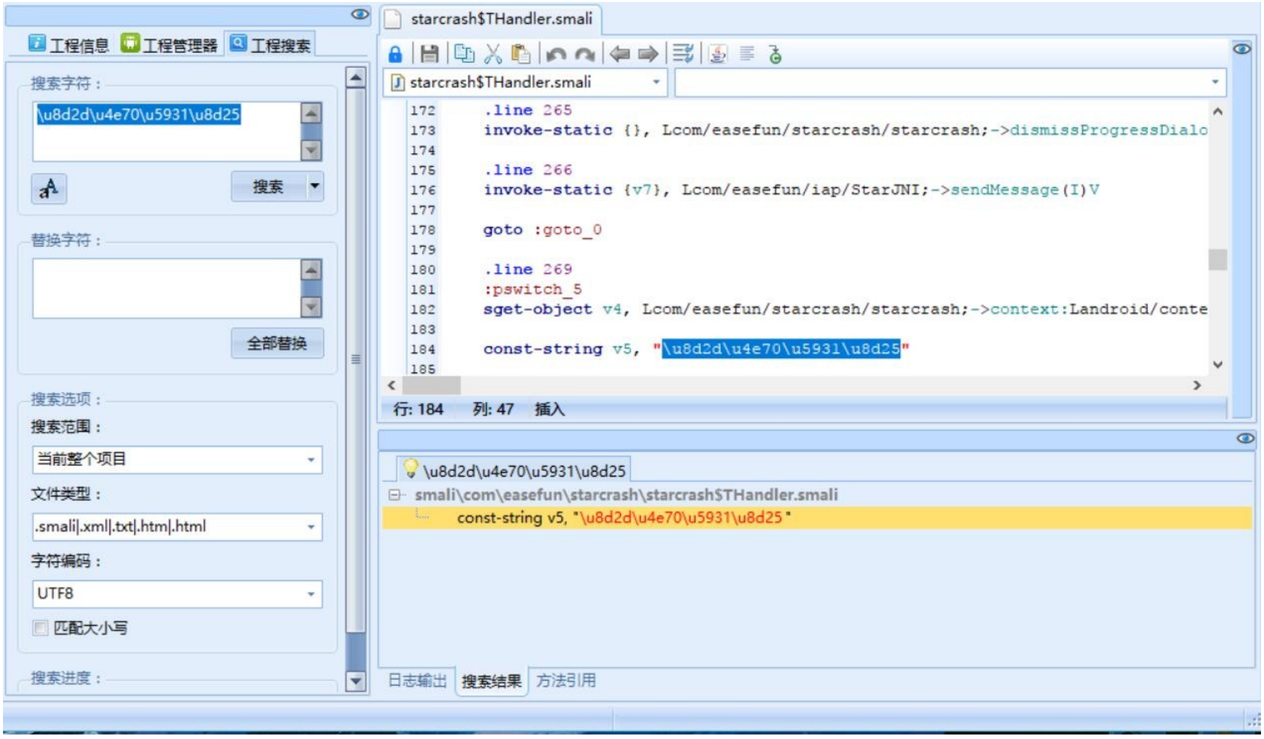
我们点击“领取”之后，会弹出“支付中，等待支付结果”。如果你是用模拟器玩的，过一会儿会弹出“购买失败”：



信息收集完毕，拖入 Android Killer：



搜索“购买失败”，上下文中应该会有“购买成功”。直接搜索原文本是没有用的，这里我就不演示了，要搜索 Unicode 编码形式 `\u8d2d\u4e70\u5931\u8d25` 。



在 `starcrash$THandler.smali` 的 `handleMessage` 方法中找到了这个文本，这个类是 `starcrash` 类中的闭包。

```
.line 269
:pswitch_5
sget-object v4, Lcom/easefun/starcrash/starcrash;->context:Landroid/content/Context;

const-string v5, "\u8d2d\u4e70\u5931\u8d25" # 购买失败

invoke-static {v4, v5, v6}, Landroid/widget/Toast;->makeText(Landroid/content/Context;Ljava/lang/CharSequence;I)Landroid/widget/Toast;

move-result-object v4

invoke-virtual {v4}, Landroid/widget/Toast;->show()V

.line 270
invoke-static {}, Lcom/easefun/starcrash/starcrash;->dismissProgressDialog()V

.line 271
invoke-static {v6}, Lcom/easefun/iap/StarJNI;->sendMessage(I)V

goto :goto_0 # return-void
```

可以看出这是 `switch` 结构的一个分支，我们到跳转表处看看：

```
.line 236
iget v4, p1, Landroid/os/Message;->what:I

packed-switch v4, :pswitch_data_0

# ...

.line 236
:pswitch_data_0
.packed-switch 0x0
    :pswitch_0
    :pswitch_1
    :pswitch_2
    :pswitch_3
    :pswitch_4
    :pswitch_5 # 购买失败
    :pswitch_6
    :pswitch_7
    :pswitch_8
    :pswitch_9
.end packed-switch
```

这个代码是安卓的跨线程消息传递机制，不懂可以直接搜索 `Handler` 。这个 `switch` 枚举了 `Message` 的 `what` 参数，该参数用于区分消息的不同种类。问题来了， `what` 值的含义是开发者自己定制的，而且外部类里面也没有相关常量。

如果不想分析代码，可以把该值改成 0~9，每个都试一遍。但是总归有不这么麻烦的办法，那就是分析代码。有两种方式，第一种是从这几个分支里面找到成功分支，第二种是从外部类的线程入口函数中找到成功的代码。我这里选前者。

我们简单遍历一下各分支的字符串吧（具体代码省略）。

分支编号	消息	行号
2	请确认SIM卡已插入	124
3	支付需要网络连接	144
4	购买成功	164
5	购买失败	184
6	购买取消	204
8	支付中，等待支付结果	255
9	已成功领取今日特权礼包中的十个钻石	269

只有这几个分支是有消息的，而且观察得出，这个 `handler` 不仅仅处理购买成功和失败消息，还处理了其它无关的消息。这种情况下就不能强行都改成第 5 个分支。我们可以考虑把第 2、3、5、6 都改成第四个分支。

```
.packed-switch 0x0
    :pswitch_0
    :pswitch_1
    :pswitch_4 # 2
    :pswitch_4 # 3
    :pswitch_4
    :pswitch_4 # 5
    :pswitch_4 # 6
    :pswitch_7
    :pswitch_8
    :pswitch_9
.end packed-switch
```

重新编译并打包后，我们试一试：


夜神安卓模拟器
关卡: 1 目标: 1000
310
0
历史最高分
0
购买成功

它这个付费是通过短信实现的，它会直接发送短信，无法自己输入手机号。所以，如果你插着电话卡玩还是会扣费的，这一点可以通过移除`AndroidManifest.xml`中的`SEND_SMS`权限来解决。

4.6 去广告

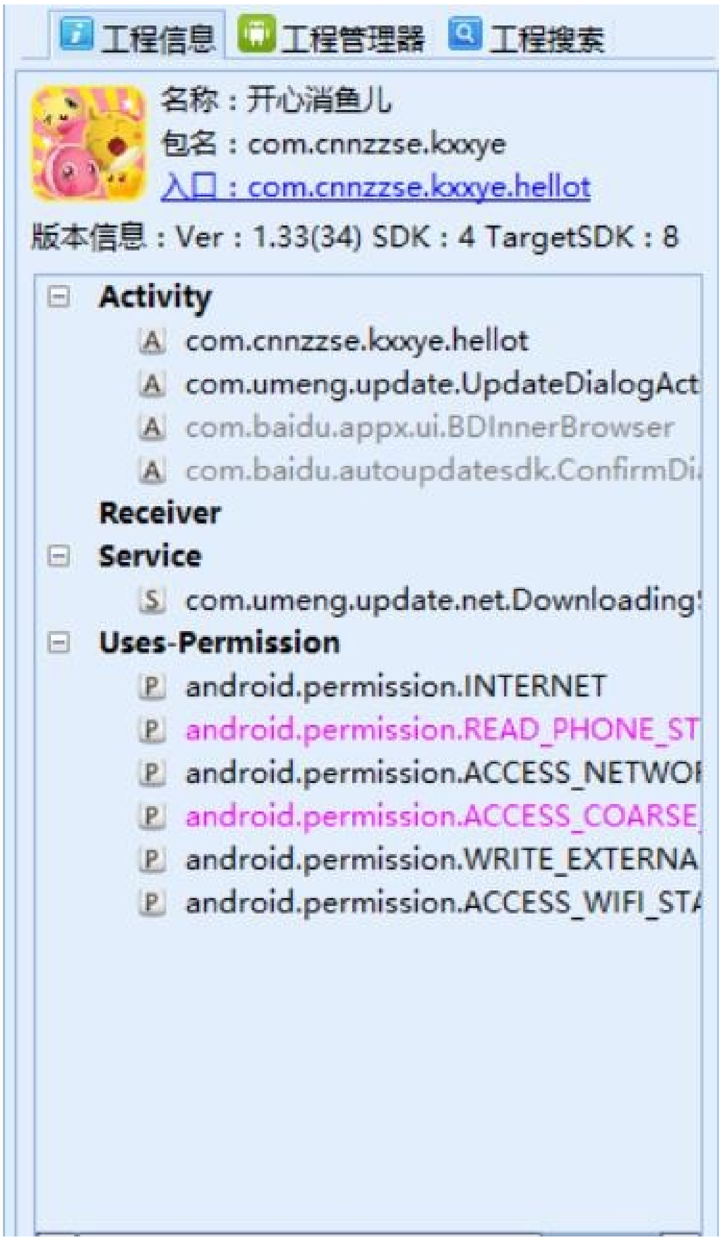
> 作者：飞龙

软件在这里下载：http://www.yxdown.com/shouji/376800.html

我们要去掉的是主界面上的广告：



把它拖进 Android Killer，这个项目的包是 `com.cnnzzse.kxxye` ，通过查询配置文件可得知，主界面是 `hellot` 。



我们在这个类中搜索 `ad` ：

```
.field private adView:Lcom/baidu/mobads/AdView;

# ...

.field private interAd:Lcom/baidu/mobads/InterstitialAd;

.field private isInitAd:Z

# ...

.field private showAd:Z

# ...

.method public showAdView()V

# ...

.method public showInterView()V

# ...
```

可以看到一共有两个广告， `adView` 和 `interAd` ，我们再来看 `showAdView` 和 `showInterView` 的方法：

```
# showAdView
.line 283
iget-boolean v1, p0, Lcom/cnnzzse/kxxye/hellot;->showAd:Z

if-nez v1, :cond_1

.line 300
:cond_0
:goto_0
return-void

# showInterView
.line 302
iget-boolean v0, p0, Lcom/cnnzzse/kxxye/hellot;->showAd:Z

if-nez v0, :cond_0

.line 346
:goto_0
return-void
```

它们都通过 `showAd` 这个字段来判断是否要显示广告。

我们可以想办法把这个字段给赋成 false 。搜索 Lcom/cnnzzse/kxxye/hellot;->showAd:Z ：



发现对其写入的一共两处。第一处是构造器 <init> 中。

```
.prologue
const/4 v1, 0x1

# ...

.line 74
const/4 v0, 0x0

# ...

.line 75
iput-boolean v1, p0, Lcom/cnnzzse/kxxye/hellot;->showAd:Z
```

这里我们把 v1 改成 v0 。

第二处是静态方法 access$0 ，这个方法专门用于设置 showAd

```
.method static synthetic access$0(Lcom/cnnzzse/kxxye/hellot;Z)V
    .locals 0

    .prologue
    .line 75
    iput-boolean p1, p0, Lcom/cnnzzse/kxxye/hellot;->showAd:Z

    return-void
.end method
```

我们加上一句 const/4 p1, 0x0 。

完事。

4.7 修改游戏金币

> 作者：飞龙

软件下载：http://www.xuepojie.com/thread-24343-1-1.html

进入游戏之后会有个“每日登录奖励”弹窗：



点击之后会看到“您获得 100 金币”：



我们看一下金币数量，`100, 150, 200, 300`。好了，将软件拖进 Android Killer：



我们搜索“您获得”，定位到了 `bu.smali` ：

```
new-instance v0, Ljava/lang/StringBuilder;

const-string v1, "\u60a8\u83b7\u5f97 " # 您获得

invoke-direct {v0, v1}, Ljava/lang/StringBuilder;-><init>(Ljava/
lang/String;)V

iget-object v1, p0, Lcom/linkstudio/FruitLink/a/bu;->ai:[I

aget v1, v1, p1

invoke-virtual {v0, v1}, Ljava/lang/StringBuilder;->append(I)Lja
va/lang/StringBuilder;

move-result-object v0

const-string v1, " \u91d1\u5e01" # 金币

invoke-virtual {v0, v1}, Ljava/lang/StringBuilder;->append(Ljava
/lang/String;)Ljava/lang/StringBuilder;

move-result-object v0
```

我们发现它是从 `ai` 数组获得数据。我们在当前文件中搜索 `Lcom/linkstudio/FruitLink/a/bu;->ai` ，发现只有一处对其赋值：

```
new-array v0, v1, [I

fill-array-data v0, :array_0

iput-object v0, p0, Lcom/linkstudio/FruitLink/a/bu;->ai:[I

# ...

:array_0
.array-data 4
    0x64  # 100
    0x96  # 150
    0xc8  # 200
    0x12c # 250
.end array-data
```

因为这个数组是 `int[]` ，我们尝试都改成 `0x7fffffff` （ `int` 的最大值）。然后保存，重新打包，运行：



另外这个游戏中还是有内购，具体破解方法不再赘述了，请参见“糖果星星达人”一节。

4.8 去广告 II

> 作者：飞龙

软件下载：http://www.xuepojie.com/thread-24545-1-1.html

事先声明，这个软件没有功能，是个壳子，我们主要研究如何去广告。

软件的主界面是这样的，可以看到最下方的“有米”：



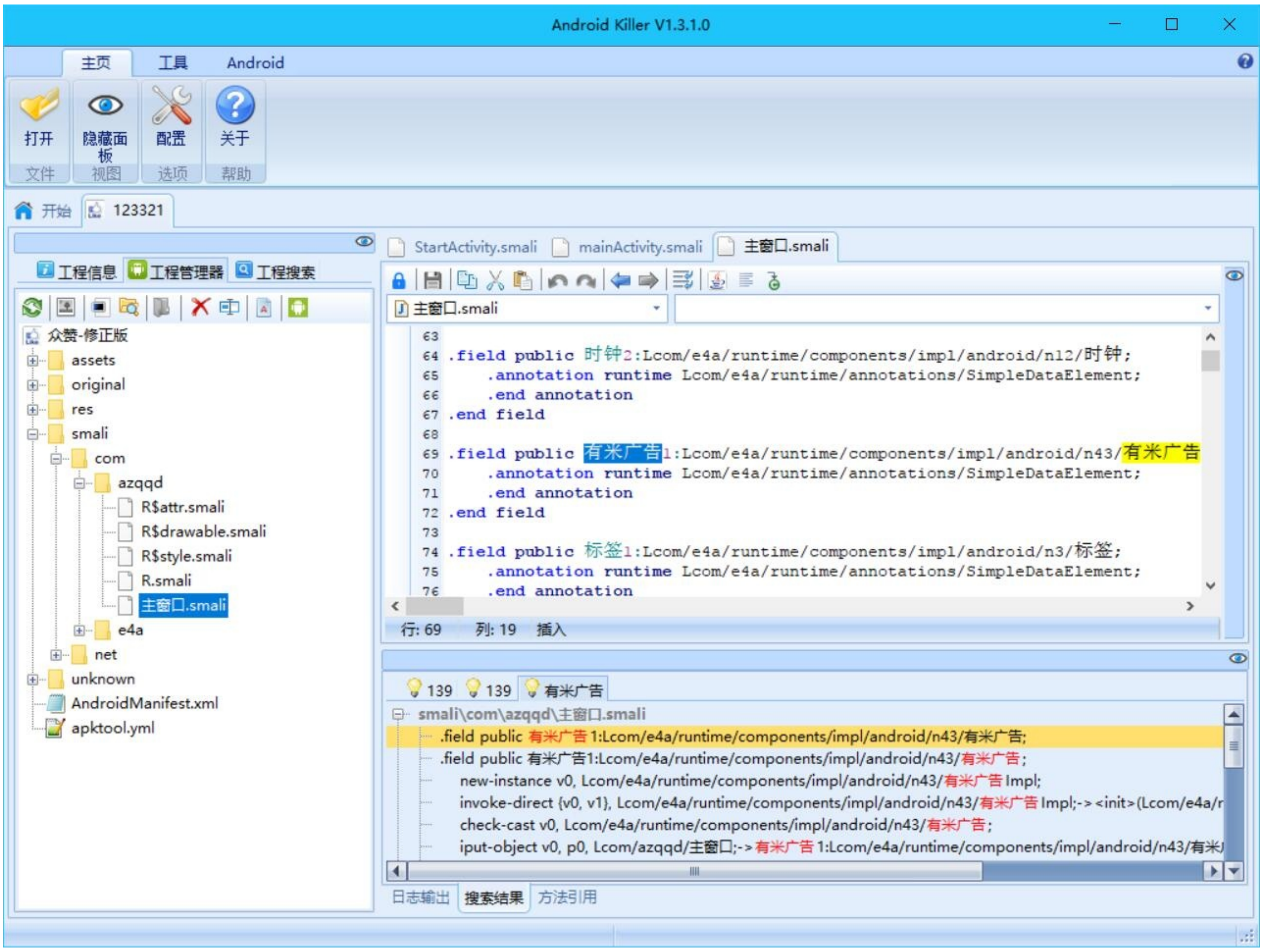
中间是一些按钮，点击每个按钮都会出现广告：



拖入 AK，发现是 e4a 编写的。


名称：众赞-修正版
包名：com.azqqd
入口：com.e4a.runtime.android.Start...
版本信息：Ver：1.0(1) SDK：8 TargetSDK：N/A
Receiver
net.youmi.android.AdReceiver
Service
net.youmi.android.AdService
net.youmi.android.ExpService
Uses-Permission
android.permission.GET_TASKS

我们搜索“有米广告”：


Android Killer V1.3.1.0
主窗口.smali
.field public 有米广告1:Lcom/e4a/runtime/components/impl/android/n43/有米广告
.annotation runtime Lcom/e4a/runtime/annotations/SimpleDataElement;
.end annotation
.end field

按照 e4a，“主窗口”应该就是主界面，`StartActicity` 只是个包装。e4a 的界面都是动态创建的，在 `$define` 方法中。

我们观察 `主窗口$创建完毕` 方法：

```
.method public 主窗口$创建完毕()V
    .locals 6

    .prologue
    const/4 v3, 0x1

    .line 12
    iget-object v0, p0, Lcom/azqqd/主窗口;->有米广告1:Lcom/e4a/run
time/components/impl/android/n43/有米广告;

    const-string v1, "80b3665dbe24da6c"

    const-string v2, "b1ec288d42c62f47"

    move v4, v3

    move v5, v3

    invoke-interface/range {v0 .. v5}, Lcom/e4a/runtime/componen
ts/impl/android/n43/有米广告;->初始化广告(Ljava/lang/String;Ljava/l
ang/String;ZZZ)V

    .line 13
    iget-object v0, p0, Lcom/azqqd/主窗口;->有米广告1:Lcom/e4a/run
time/components/impl/android/n43/有米广告;

    invoke-interface {v0}, Lcom/e4a/runtime/components/impl/andr
oid/n43/有米广告;->显示插播广告()V

    .line 14
    iget-object v0, p0, Lcom/azqqd/主窗口;->有米广告1:Lcom/e4a/run
time/components/impl/android/n43/有米广告;

    const-string v1, "\u4f17\u8d5e-\u4fee\u6b63\u7248"

    invoke-interface {v0, v1}, Lcom/e4a/runtime/components/impl/
android/n43/有米广告;->设置积分墙标题(Ljava/lang/String;)V

    .line 15
    iget-object v0, p0, Lcom/azqqd/主窗口;->有米广告1:Lcom/e4a/run
time/components/impl/android/n43/有米广告;

    invoke-interface {v0}, Lcom/e4a/runtime/components/impl/andr
oid/n43/有米广告;->显示插播广告()V

    return-void
.end method
```

直接在这个方法的开头插入 `return-void` ，就没了。

之后是按钮的广告，由于按钮太多，一共有八个，我这里仅仅演示左上角的按钮（139 那个）。

在当前文件中搜索`139`：

```
iget-object v0, p0, Lcom/azqqd/主窗口;->按钮1:Lcom/e4a/runtime/com
ponents/impl/android/n1/按钮;

const-string v1, "139\u79ef\u5206500\u8d5e"

invoke-interface {v0, v1}, Lcom/e4a/runtime/components/impl/andr
oid/n1/按钮;->标题(Ljava/lang/String;)V
```

得知它就是按钮 1。

然后找到`按钮1$被单击`方法：

```
.method public 按钮1$被单击()V
    .locals 1

    .prologue
    .line 26
    const-string v0, "\u8bf7\u5148\u5b8c\u6210\u5206\u4eabQQ\u7f
a4\u4efb\u52a1"

    invoke-static {v0}, Lcom/e4a/runtime/应用操作;->弹出提示(Ljava/
lang/String;)V

    .line 27
    iget-object v0, p0, Lcom/azqqd/主窗口;->有米广告1:Lcom/e4a/run
time/components/impl/android/n43/有米广告;

    invoke-interface {v0}, Lcom/e4a/runtime/components/impl/andr
oid/n43/有米广告;->显示插播广告()V

    return-void
.end method
```

可以看到它没有任何实际功能，直接在开头插入`return-void`。我们之后只需要堆其余 7 个按钮执行相同操作就好了。

4.9 破解内购 II

> 作者：飞龙

这次要破解的游戏是这个：http://dl.pconline.com.cn/download/544623.html

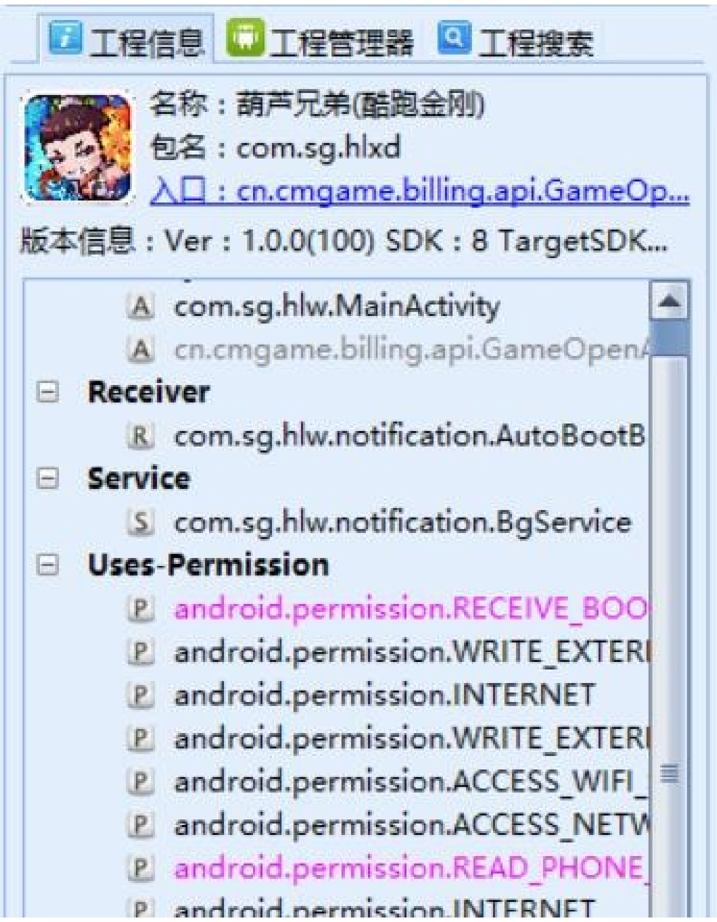
打开游戏后，主界面有个 VIP，我们要破解的就是这个东西：



我们切换到“好友代付”，点击“购买”，然后是“确认支付”：



提示“购买道具xxx失败”。好，我们载入 AK：



我们搜索“失败”的 Unicode，在 `SDKMessage$1` 中找到了失败的文本。

```
.line 203
:pswitch_1
new-instance v1, Ljava/lang/StringBuilder;

# 购买道具：[
const-string v2, "\u8d2d\u4e70\u9053\u5177\uff1a["

invoke-direct {v1, v2}, Ljava/lang/StringBuilder;-><init>(Ljava/
lang/String;)V

invoke-virtual {v1, p2}, Ljava/lang/StringBuilder;->append(Ljava
/lang/String;)Ljava/lang/StringBuilder;

move-result-object v1

# ] 失败！
const-string v2, "] \u5931\u8d25\uff01"

invoke-virtual {v1, v2}, Ljava/lang/StringBuilder;->append(Ljava
/lang/String;)Ljava/lang/StringBuilder;

move-result-object v1

invoke-virtual {v1}, Ljava/lang/StringBuilder;->toString()Ljava/
lang/String;

move-result-object v0
```

这是 `switch` 是一个分支，我们直接往上找：

```
.line 197
.local v0, "result":Ljava/lang/String;
packed-switch p1, :pswitch_data_0

# ...

.line 197
:pswitch_data_0
.packed-switch 0x1
    :pswitch_0
    :pswitch_1
.end packed-switch
```

按照惯例，我们把跳转表都改成 `:pswitch_0` ，编译，打包，安装。然后随便购买一个东西。虽然 SDK 的对话框提示失败，这个我们改不了，但是游戏的 `Toast` 提示成功。


夜神安卓模拟器
通关抽奖
获得150钻,5000金,10开局冲刺,5护盾,5终极冲刺
x5
x5000
x10
x50
终极冲刺
x5
钻石
x100
大娃碎片
x1
生命接力
x5
购买道具：[012] 成功！

4.10 玄奥八字

> 作者：飞龙
>
> 声明：本人极度厌恶玄学，选取此软件是为了研究逆向技术，并不代表本人赞成其内容。

这次要破解的软件是这个：http://www.xazhouyi.com/android/soft/bazi.html

首先分析其行为，打开软件：



按照以往的经验，程序有个字段用于维护注册状态，我们可以通过字符串快速定位到它。我们将其载入 AK：



搜索“软件未注册”，在 `string.xml` 中找到：

```
<string name="Id_StartInfo">"注意事项
1：软件未注册只能使用1999年的，注册后可使用所有功能。
2：注册方法请看软件菜单“帮助->软件帮助”。
3：注册时需提供软件系列号，软件系列号--点击菜单“帮助->注册”可看到。
...
</string>
```

然后在 `public.xml` 中找到，字符串的 ID 是 `0x7f060003` 。之后搜索这个数字。

我们在 `main` ，也就是入口的 `MyInit` 函数中找到了这个数值：

```
:cond_b
# 玄奥八字7.2未注册！
const-string v7, "\u7384\u5965\u516b\u5b577.2\u672a\u6ce8\u518c\
uff01"

invoke-virtual {p0, v7}, LMy/XuanAo/BaZiYi/main;->setTitle(Ljava
/lang/CharSequence;)V

.line 199
invoke-virtual {p0}, LMy/XuanAo/BaZiYi/main;->getResources()Land
roid/content/res/Resources;

move-result-object v7

const v8, 0x7f060003

invoke-virtual {v7, v8}, Landroid/content/res/Resources;->getStr
ing(I)Ljava/lang/String;

# 刚才的字符串
move-result-object v7

invoke-static {p0, v7, v10}, Landroid/widget/Toast;->makeText(La
ndroid/content/Context;Ljava/lang/CharSequence;I)Landroid/widget
/Toast;

move-result-object v7

invoke-virtual {v7}, Landroid/widget/Toast;->show()V

goto :goto_4 # return-void
```

我们猜想 `:cond_b` 是失败分支，我们往上找：

```
sget-object v7, LMy/XuanAo/BaZiYi/main;->m_chkSoft:LMy/XuanAo/BaZiYi/CSoftReg;

invoke-virtual {v7}, LMy/XuanAo/BaZiYi/CSoftReg;->ChkNumA()Z

move-result v7

if-eqz v7, :cond_b

sget-object v7, LMy/XuanAo/BaZiYi/main;->m_chkSoft:LMy/XuanAo/BaZiYi/CSoftReg;

invoke-virtual {v7}, LMy/XuanAo/BaZiYi/CSoftReg;->ChkNumB()Z

move-result v7

if-eqz v7, :cond_b

sget-object v7, LMy/XuanAo/BaZiYi/main;->m_chkSoft:LMy/XuanAo/BaZiYi/CSoftReg;

invoke-virtual {v7}, LMy/XuanAo/BaZiYi/CSoftReg;->ChkNumC()Z

move-result v7

if-eqz v7, :cond_b # 关键跳

# 成功分支

.line 195
const-string v7, "\u7384\u5965\u516b\u5b57.2"

invoke-virtual {p0, v7}, LMy/XuanAo/BaZiYi/main;->setTitle(Ljava/lang/CharSequence;)V

.line 196
# const/4 v10, 0x1
sput-boolean v10, LMy/XuanAo/BaZiYi/main;->m_regFlag:Z
```

我们可以得出 `ChkNumA/B/C` 是三个关键判断。下面的 `if-eqz` 是关键跳。成功各分支将 `m_regFlag` 设为 1，说明它是保存注册状态的字段。

我们可以将这三个 `if-eqz` 都给注释掉，但是我们可以采取另一种方式，在最后一个 `if` 的下面添加 `:goto_100` 标签，然后在第一个 `if` 上面添加 `goto :goto_100` 。

```
.line 193
:cond_1
goto :goto_100
sget-object v7, LMy/XuanAo/BaZiYi/main;->m_chkSoft:LMy/XuanAo/BaZiYi/CSoftReg;

# ...

if-eqz v7, :cond_b
:goto_100
```

重新打包、安装软件后，打开软件，我们发现不再弹出注册提示了。访问菜单->更多->注册之后，在注册界面中我们可以看到“已注册”。



4.11 优酷 APK 去广告

> 作者：飞龙

软件下载：http://app.cnmo.com/android/235159/

这次要破解优酷的 APK，去掉播放视频开头的广告：



我们先抓包，看到了 `api.mobile.youku.com`，这个就是广告所在的域名。



128	200	HTTP	r4.ykimg.com	/05100000577 6FD176714C0722302BA99	283,240	max-ag...	ir
129	200	HTTP	a.play.api.3g.youku.com	/common/v3/play?_t_=1467424516&e=md...	15,809	max-ag...	a
130	200	HTTP	api.mobile.youku.com	/layout/android5_0/play/detail?pid=7613b...	1,926	max-ag...	a
131	200	HTTP	count.atm.youku.com	/mlog?lvs=2&sp=0&st=1&bt=1&os=1&avs...	0	private...	b
132	200	HTTP	ad.api.3g.youku.com	/adv?_t_=1467424517&e=md5&_s_=69d...	15,590	max-ag...	a
133	200	HTTP	api.mobile.youku.com	/shows/21073c9078a111e5b692/reverse/...	1,109	max-ag...	a
134	200	HTTP	api.mobile.youku.com	/adv/banner?site=1&p=1433218285&ac=...	1,475	max-ag...	a

我们载入 AK：



将所有 `api.mobile.youku.com` 都换成 `127.0.0.1` 即可：



然后，在回编译的时候，会有如下问题。

```
>D:\Wizard破解工具包\Tool\Android\AndroidKiller_v1.3.1\projects\Youku\Project\res\values-v23\styles.xml:6: error: Error retrieving parent for item: No resource found that matches the given name '@android:style/WindowTitleBackground'.
>D:\Wizard破解工具包\Tool\Android\AndroidKiller_v1.3.1\projects\Youku\Project\res\values-v23\styles.xml:6: error: Error retrieving parent for item: No resource found that matches the given name '@android:style/WindowTitleBackground'.
```

我们找到 `res/value-v23/styles.xml` ，把 `resources` 下的东西注释掉：

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <!--
    ...
    -->
</resources>
```

在找到 `res/value/public.xml` ，把所有带 `Base.V23` 的东西（两个）注释掉：

```
<!--
<public type="style" name="Base.V23.Theme.AppCompat" id="0x7f0d00a6" />
<public type="style" name="Base.V23.Theme.AppCompat.Light" id="0x7f0d00a7" />
-->
```

即可成功编译：

```
当前 Apktool 使用版本：Android Killer Default APKTOOL
正在编译 APK，请稍等...
>I: 使用 ShakaApktool 2.0.0-20150914
>I: 编译 smali 到 classes.dex...
>I: 编译 smali_classes2 到 classes2.dex...
>I: 正在编译资源...
>I: 正在拷贝libs目录... (/lib)
>I: 正在编译apk文件...
>I: 复制未知文件/目录...
APK 编译完成!
正在对 APK 进行签名，请稍等...
APK 签名完成!
-----------------------------
APK 所有编译工作全部完成!!!
生成路径：
file:D:\Wizard破解工具包\Tool\Android\AndroidKiller_v1.3.1\projects\Youku\Bin\Youku_killer.apk
```

4.12 MagSearch 1.8 爆破

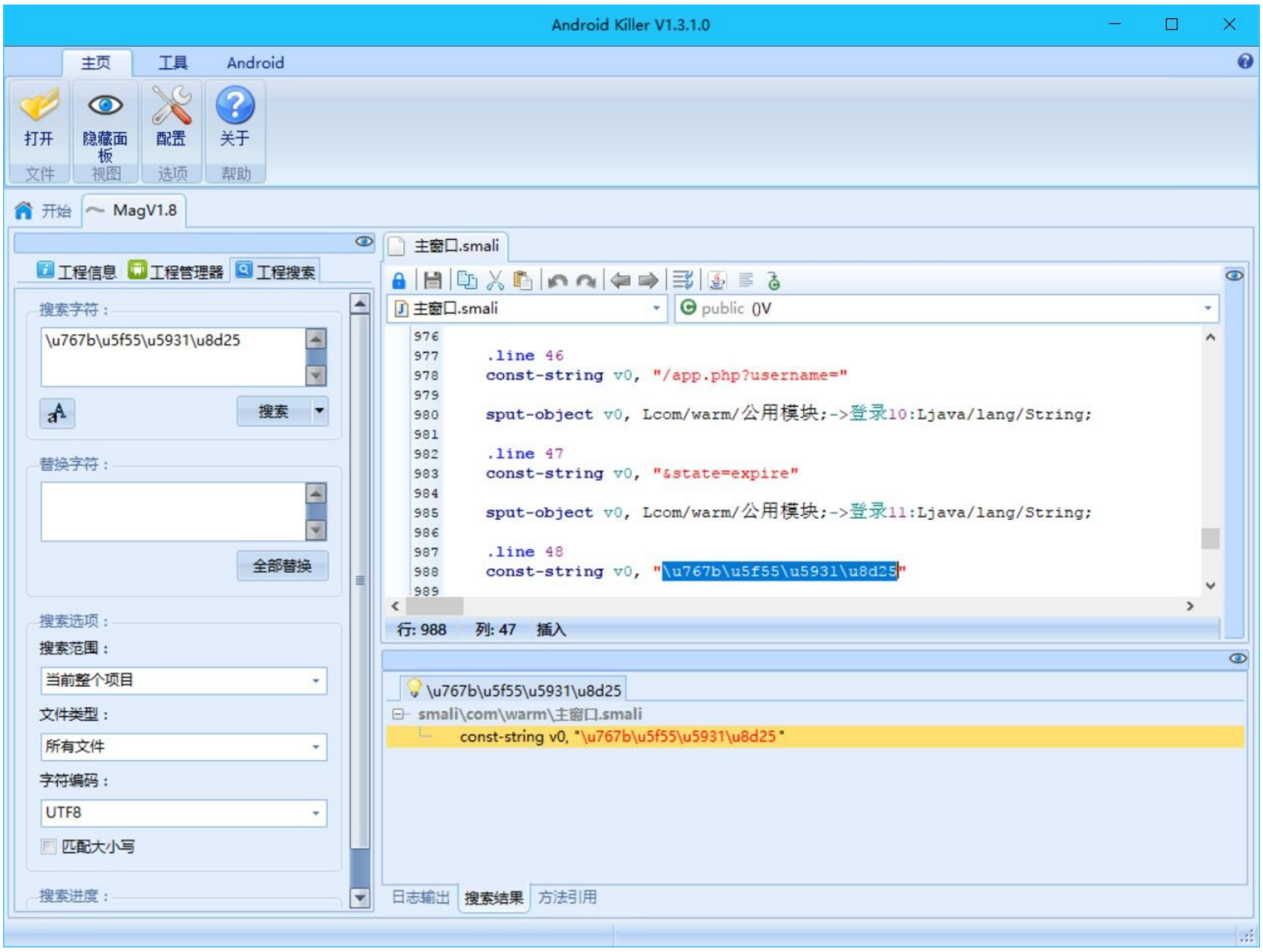
> 作者：飞龙

软件下载：http://www.xuepojie.com/thread-26549-1-1.html

打开之后是登录界面，随便输入用户名和密码，显示登录失败。



将其载入 AK，搜索“登录失败”，没有反应。搜索它的 Unicode 编码 `\u767b\u5f55\u5931\u8d25` ：

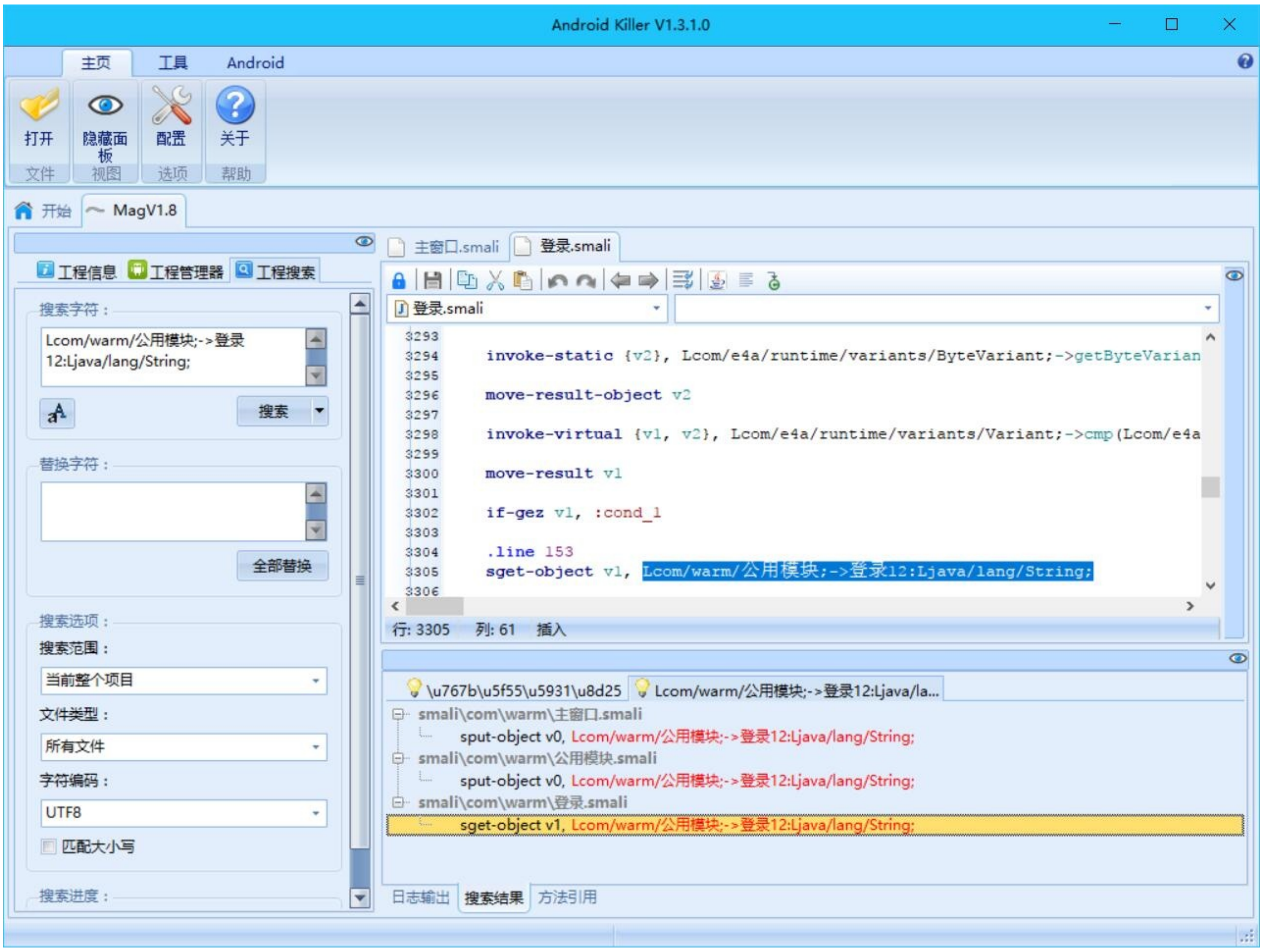


在主窗口的 `时钟1$周期事件` 中，出现了：

```
.line 48
const-string v0, "\u767b\u5f55\u5931\u8d25" # 登录失败

sput-object v0, Lcom/warm/公用模块;->登录12:Ljava/lang/String;
```

我们搜索 `Lcom/warm/公用模块;->登录12:Ljava/lang/String;` ：



根据结果，登录窗口中（ `登录.smali` ）使用了该字段。

```
if-gez v1, :cond_1

.line 153
sget-object v1, Lcom/warm/公用模块;->登录12:Ljava/lang/String;

invoke-static {v1}, Lcom/e4a/runtime/应用操作;->弹出提示(Ljava/lang
/String;)V

:cond_1
return-void
```

我们找到调用该字段的地方，这个应该是失败分支，需要向上找到关键判断。

```
.line 152
:cond_0
# 失败分支

# ...

if-eqz v1, :cond_0

# 成功分支
# ...
```

我们向上找到 `:cond_0` ，如果不跳到这里，就能走成功各分支。我们接着寻找谁使用了 `:cond_0` ，然后找到了关键判断。

我们如果想要爆破，一个思路就是把这个关键判断注释掉。但是这样还是要经过这个登录窗口，不够美观。我们现在换一个思路，如果登录窗口不是主窗口，那么我们只需要找到主窗口启动登录窗口的地方，把它改成登录后的那个窗口，就行了。

我们在成功分支中找到：

```
.line 149
# 热门资源
sget-object v1, Lcom/warm/公用模块;->登录9:Ljava/lang/String;

invoke-static {v1}, Lcom/e4a/runtime/应用操作;->读取窗口(Ljava/lang
/String;)Lcom/e4a/runtime/components/impl/android/窗口Impl;

move-result-object v1

check-cast v1, Lcom/e4a/runtime/components/窗口;

invoke-static {v1}, Lcom/e4a/runtime/应用操作;->切换窗口(Lcom/e4a/r
untime/components/窗口;)V
```

也就是说，登录窗口之后是热门资源。我们回到 `时钟1$周期事件` ，找到启动登录窗口的代码：

```
.line 70
const-string v0, "\u767b\u5f55" # 登录

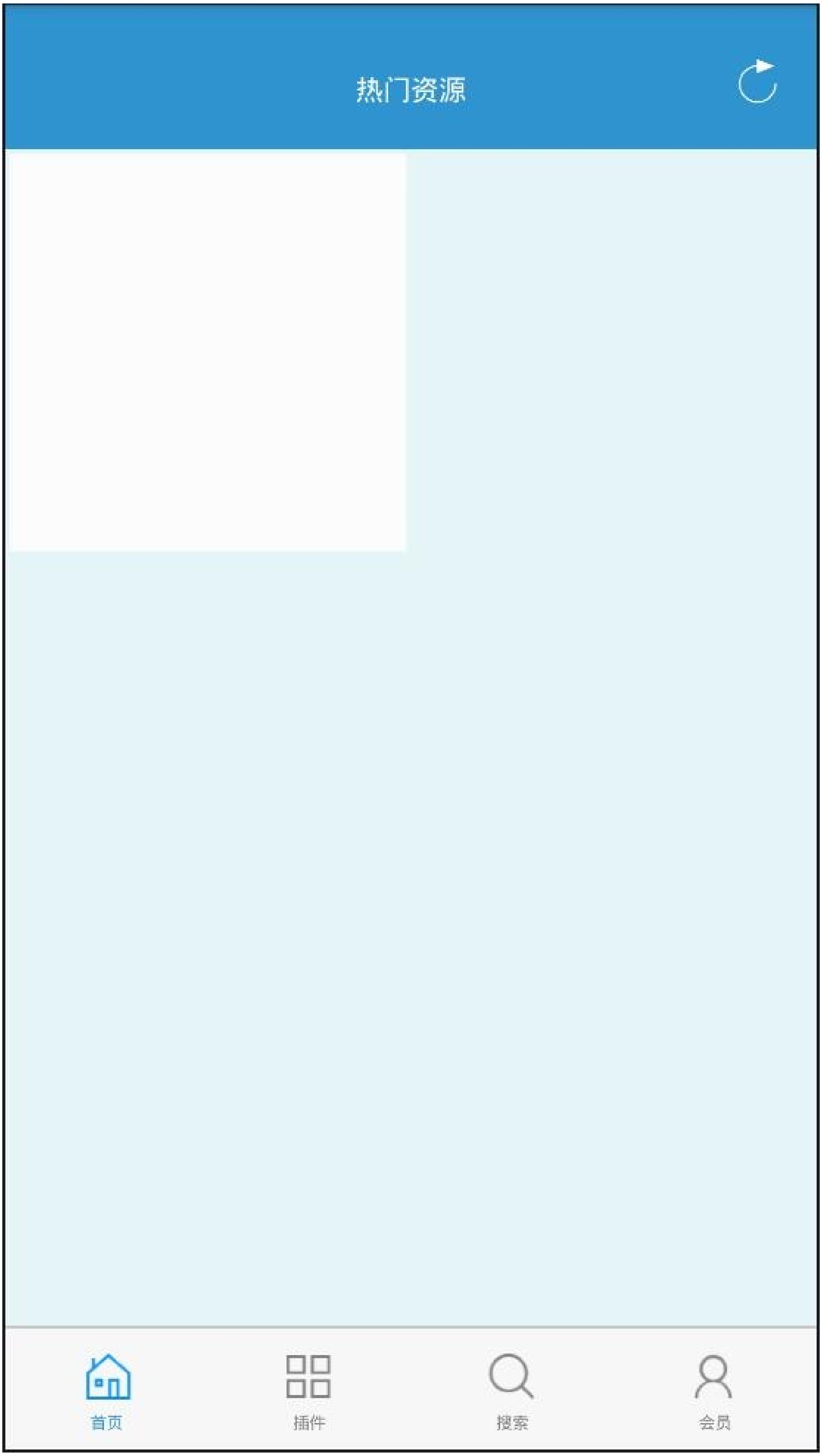
invoke-static {v0}, Lcom/e4a/runtime/应用操作;->读取窗口(Ljava/lang/String;)Lcom/e4a/runtime/components/impl/android/窗口Impl;

move-result-object v0

check-cast v0, Lcom/e4a/runtime/components/窗口;

invoke-static {v0}, Lcom/e4a/runtime/应用操作;->切换窗口(Lcom/e4a/runtime/components/窗口;)V
```

把最上面那个字符串改成 `\u70ed\u95e8\u8d44\u6e90` （热门资源），编译打包安装之后试试。


热门资源
首页
插件
搜索
会员

成功跳过了登录页面。不过这软件现在已经废了，只能用来练手了。